Hub para comunicação digital

João Paulo Marques Reis

Dissertação para a obtenção do Grau de Mestre em

Engenharia Electrotécnica e de Computadores

Orientador: Prof. João Nuno de Oliveira e Silva

Júri:

Presidente: Prof. António Manuel Raminhos Cordeiro Grilo

Orientador: Prof. João Nuno de Oliveira e Silva

Vogal: Prof. Fernando Henrique Côrte-Real Mira da Silva

Novembro de 2018

Declaração

Declaro que o presente documento é um trabalho original da minha autoria e que cumpre todos os requisitos do código de conduta e boas práticas da Universidade de Lisboa.

Resumo

Muitas organizações usam um sistema POTS em paralelo com um sistema VoIP, isto cria um problema de fiabilidade porque se o hardware que suporta o sistema falhar a reparação imediata pode ser muito difícil dada a escassez de peças no mercado. Posto este problema este trabalho tem como objectivo oferecer um sistema de reserva que permita á organização continuar o seu normal funcionamento e oferecer funcionalidades adicionais como uma extensão do sistema existente.

Este problema pode ser resolvido utilizando um sistema que integre a tecnologia de comunicação WebRTC com as tecnologias VoIP oferecendo aos seus futuros utilizadores um sistema que lhes permita realizar chamadas através de uma aplicação Web sem necessidade de instalar qualquer tipo de código nativo.

O sistema é composto por um servidor Web, um PBX Asterisk e um servidor IVR, o servidor Web é utilizado para distribuir uma aplicação Web, como servidor de sinalização para as comunicações WebRTC entre browsers e também para configurar o servidor Asterisk com os dados dos utilizadores. O servidor PBX Asterisk é usado para interligar a aplicação Web, distribuída pelo servidor, e os telefones VoIP existentes, por fim o servidor IVR oferece menus IVR dinâmicos escritos em NodeJS permitindo uma total integração com os serviços da Web.

Com este sistema os utilizadores conseguem realizar todas as operações que um telefone real permite e além dessas conseguem procurar pessoas pelo seu nome podendo ligar assim que as encontrem e ainda rever as gravações que fazem antes do enviar.

Palavras-chave: Voz sobre IP, WebRTC, Asterisk, NodeJS, IVR, Desenvolvimento Web

Abstract

Many organizations use a POTS system in conjunction with a VoIP, this creates an availability problem to system because the repair and replacement of the hardware that keeps the POTS system working is very hard due to lack of spare parts on the market. Given this problem this work has the objective of offering a backup system that allows the organization that implements it to keep functioning as normal and at the same time offering extra features working as an extension of the existing system.

This problem can be solved using a system that integrates the new webRTC technology with the existing VoIP technologies offering the future users of the system a Web application that allows them to do calls without the need of any native code.

The system is composed of a Web server, an Asterisk PBX and an IVR server, the Web server is used to deliver a WebApp, signaling server for WebRTC browsers and to set up users inside the Asterisk server. The Asterisk PBX is used to connect the WebApp to the already existing SIP infrastructure and the IVR server serves dynamic IVR menus to the users, the menus are written using NodeJS which makes them easy to integrate with the webservices.

With this system the users can do all the operations that a real pone allows inside a browser, search other users by name, review messages before sending and see all voicemail inside the browser.

Keywords: Voice over IP, WebRTC, Asterisk, NodeJS, Web development, IVR

Índice

Lista de Figuras

Lista de tabelas

Lista de abreviações

API – Application Programming Interface

AVPF – Audio-Visual Profile Feedback

DTLS – Datagram Transport Layer Security

DTMF – Dual-Tone Multi-Frequency

HTML – Hypertext Markup Language

HTTP – Hypertext Transfer Protocol

HTTPS – Hypertext Transfer Protocol Secure

ICE – Interactive Connectivity Establishment

IP – Internet Protocol

IVR – Interactive Voice Response

JSON – JavaScript Object Notation

NAS – Network Atached Storage

NAT – Network Address Translation

P2P – Peer to Peer

PBX – Phone Branch exchange

POTS – Plain Old Telephone System

PSTN – Public Switched Telephone Network

QoS – Quality of Service

RDIS – Redes Digitais com Integração de Serviços

REST – Representational State Transfer

RTP – Real-Time Transport Protocol

SDP – Session Description Protocol

SHA – Secure Hash Algorithm

SIP – Session Initiation Protocol

SRTP – Secure Real-Time Transport Protocol

SSL – Secure Socket Layer

STUN – Session Transversal Utilities for NAT

TLS – Transport Layer Security

TURN – Transversal Using Relays around NAT

UDP – User Datagram Protocol

URI – Uniform Resource Identifier

URL – Uniform Resource Location

VOD – Video On Demand

VoIP – Voice over IP

WS – WebSocket

WSS – WebSocket Secure

1 Introdução

1.1 Enquadramento

Com a expansão da internet e o aparecimento da tecnologia Voice over IP (VoIP) as antigas redes telefónicas analógicas e digitais começaram rapidamente a ficar obsoletas. A incapacidade de competir com a qualidade sonora da nova tecnologia digital bem como os novos serviços oferecidos pelas redes VoIP. Incapazes de competir as redes telefónicas tradicionais começaram a ser lentamente substituídas pelas novas redes VoIP. Infelizmente a substituição das antigas redes telefónicas analógicas, denominadas de Plain Old Telephone System (POTS), é um processo dispendioso visto que é necessário adquirir novas cabelagens e equipamento e além disso é necessário para instalar toda a nova infraestrutura. Para minimizar este problema foram encontradas soluções híbridas em que os a rede VoIP e a rede POTS se interligam usando hardware dedicado para o efeito, permitindo assim que as duas coexistam enquanto a rede VoIP lentamente substitui a rede POTS.

1.2 Problema

Neste momento muitas empresas e organizações usam duas tecnologias em simultâneo, uma rede POTS (Plain Old Telephone Sertvice) e uma rede VoIP. A rede POTS está geralmente ligada á rede VoIP através de uma central electrónica o que lhe interagir com um Private Branch Exchange PBX.

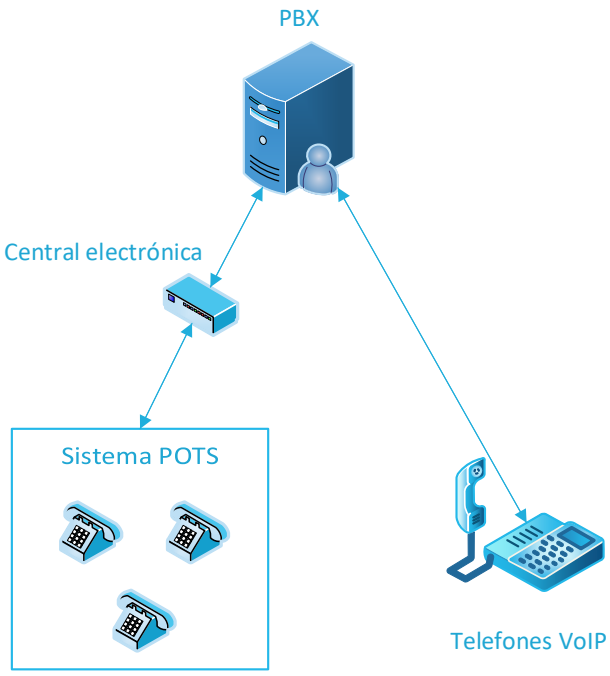

Figura 1 - Esquema da rede VoIP com POTS

A tipologia de rede da Figura 1 permite que os sistemas POTS e VoIP se interagirem como auxílio de uma central electrónica e de um Private Branch Exchanger, permitindo que os utilizadores realizem chamadas nos dois sentidos como se fosse um sistema homogéneo. No entanto isto só é possível com

a central electrónica, uma peça de hardware que nos dias que correm começa a ser cada vez mais difícil de substituir em caso de falha. Assim a manutenção das centrais eletrónicas tornou-se um desafio sendo muito difícil substituir componentes que deixem de funcionar.

Em caso de falha trocar toda a rede POTS por uma rede VoIP seria uma tarefa impossível de realizar num curto espaço de tempo o que levaria a uma falha na disponibilidade do serviço por longo período. Além do tempo que o sistema estaria em baixo o custo também seria elevado uma vez que teriam de ser adquiridos terminais VoIP, comutadores e cabos para interligar todos os terminais.

Assim torna-se importante desenvolver um sistema alternativo que seja económico e que possa funcionar em paralelo com os existentes para que em caso de necessidade possa ser usado para substituir os terminais da rede POTS.

1.3 Solução

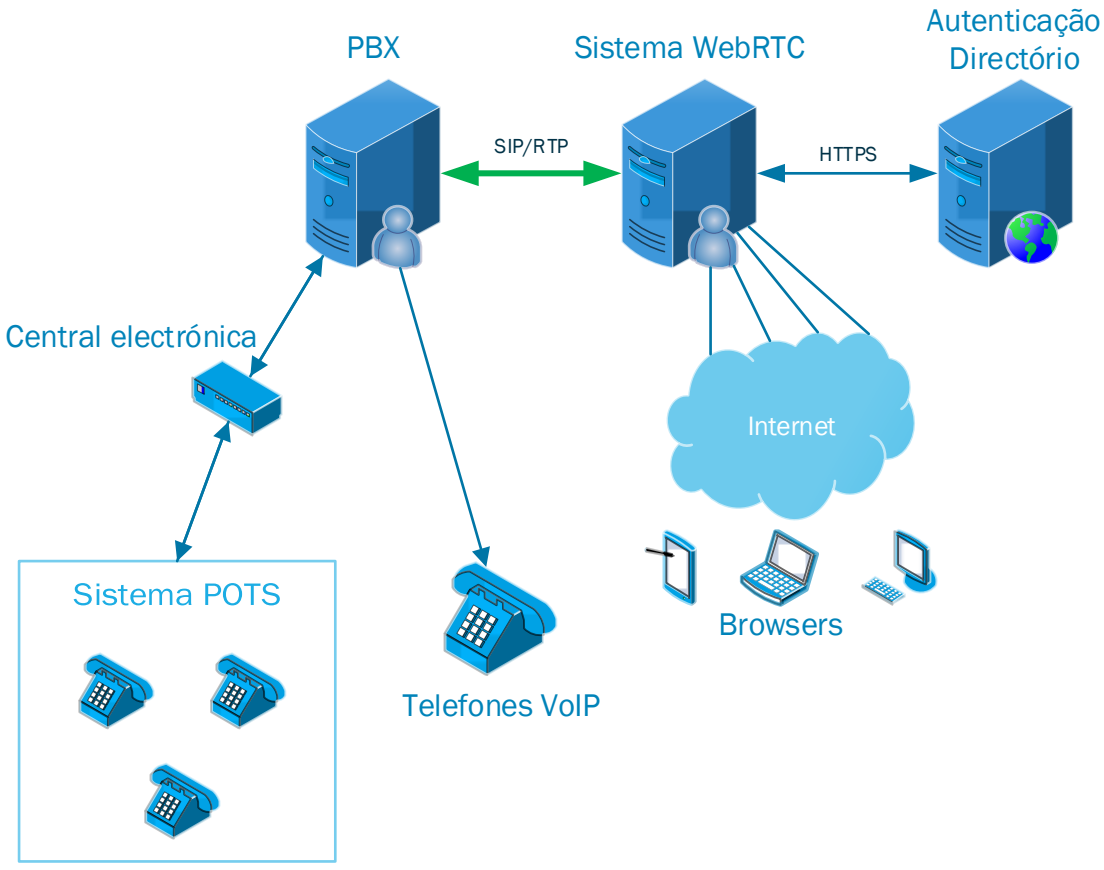

Figura 2 – Arquitectura genérica da solução do problema

A forma mais fácil de resolver o problema integrar dispositivos e redes existentes para realizar as comunicações, uma opção será utilizar a rede Internet como meio de transporte e browsers como terminais como se pode ver na Figura 2. Desta forma será possível que o sistema alcance uma grande variedade de dispositivos que vão desde os tradicionais computadores de secretária até aos telemóveis.

Isto é possível usando uma tecnologia emergente denominada de WebRTC que permite a um browser capturar áudio e vídeo através dos dispositivos ligados ao computador. Além disto também padroniza uma forma de transmitir o áudio e vídeo capturados através da internet permitindo a interligação directa com outros browsers ou com os sistemas VoIP por intermédio de um PBX.

Assim usando WebRTC é possível criar uma WebApp que realize todas as funções que um telefone VoIP realiza e ainda adicionar novas funcionalidades como a procura de contactos por nome, comunicação directa entre browsers permitindo videochamadas, e por fim centralizar todas as mensagens recebidas do voice-mail na aplicação permitindo ouvir/ver as mensagens recebidas e apagá-las se o utilizador desejar.

Com a aplicação Web do sistema será possível realizar chamadas para outros browsers de forma directa suportando videochamadas. A WebApp permite ainda que os utilizadores liguem para softohines e telefones da rede POTS com a intermediação do PBX do sistema WebRTC que comunica com o PBX do sistema existente através do protocolo SIP.

Também é possivel á aplicação receber chamadas vindas da rede POTS isto porque todos os utilizadores do sistema WebRTC são registados no PBX com um nome de utilizador e password únicos permitindo assim que estes possam ser contactados pelo PBX.

1.4 Organização do documento

O corpo deste relatório está dividido nos seguintes capítulos:

- Estado da arte / Trabalho relacionado – Descreve várias tecnologias e protocolos que usados nos sistemas de comunicação actuais, também analisa algumas das ofertas existentes no mercado para este tipo de serviço
- Sistema – Lista os requisitos, descreve a arquitectura e caracteriza os módulos que irão compor o sistema explicando o que cada um é responsável por fazer.
- Implementação – Descreve detalhadamente a implementação do sistema, listando e descrevendo as tecnologias/bibliotecas que foram utilizadas, descreve o processo de instalação e configuração dos servidores necessários. Também explica detalhadamente como é efectuada a comunicação entre os componentes de modo a cumprir os requisitos.
- Funcionamento/Validação – Explica de forma detalhada o funcionamento e utilização da interface da aplicação Web. As considerações de segurança que foram tidas durante o projecto também são abordadas nesta secção.
- Segurança – Aqui é feito um levantamento dos protocolos de segurança que foram utilizados para proteger as várias fases das comunicações.
- Discussão – Aqui é feito um balanço sobre o que poderia ter sido feito de forma diferente e do que poderá ser ainda feito de modo a melhorar o sistema que foi implementado.
- Conclusão – Faz um balanço sobre o cumprimento do dos objectivos e sobre as vantagens e desvantagens do uso do sistema.

2 Estado da arte / Trabalho relacionado

Para melhor compreender os componentes dos sistemas de comunicações actuais foi realizado um trabalho de pesquisa com vista a identificar as tecnologias e protocolos que são usados nos sistemas actuais e as tecnologias emergentes que farão parte dos sistemas do futuro. Este trabalho focou-se em três grupos de tecnologias, a telefonia antiga também conhecida como redes POTS, as actuais redes VOIP e produtos/serviços que o mercado oferece para integrar serviços VoIP com a Web.

2.1 Estado da telefonia

POTS ou Plain Old Telephone Service é o antigo serviço de telefone analógico que funcionava através de um par de fios de cobre. Este sistema necessita de um circuito para cada chamada telefónica pelo que a rede é muito pouco eficiente e dispendiosa. Os sinais de voz que são transmitidos através deste sistema são normalmente restringidos a um intervalo entre os 300 Hz e os 3300 Hz devido a limitações da largura de banda do canal, isto está muito longe da largura de banda audível que vai dos 20 Hz aos 20000 Hz. Embora a maior parte das frequências audíveis sejam eliminadas durante a transmissão o sistema ainda confiável para comunicações de voz uma vez que as frequências produzidas pelos seres humanos são muito mais limitadas do que as que podem ser ouvidas e assim é possível transmitir voz com as limitações de banda o POTS. Inicialmente as chamadas feitas usando as redes telefónicas analógicas [1] eram feitas através de um operador que estabelecia manualmente uma ligação entre os terminais. As ligações manuais viriam a partir de 1923 a ser substituídas por novos sistemas mecânicos que estabeleciam a maior parte das ligações de forma automática aumentando de forma significativa a eficiência do sistema e melhorando a experiência do utilizador.

Muito mais tarde em 1988 viriam a aparecer as redes RDIS [2] (Redes Digitais com Integração de Serviços), as redes RDIS operavam usando a infra-estrutura telefónica e permitiam realizar chamadas de voz e serviços de dados como o fax. Além disto as redes RDIS permitiam ainda a comunicação com redes de comutação de pacotes, sendo que esta última característica seria o que veria a tornar a tecnologia popular em alguns países, que acabaram por usar as redes RDIS para ligar os utilizadores á Internet uma rede puramente de dados que começava a ganhar popularidade.

Com o desenvolvimento das redes de comutação de pacotes que se focam puramente em dados e com o aparecimento de politicas de Quality of Service (QoS), que permitem as redes de comutação de pacotes dar prioridade a aplicações em tempo real a rede telefónica tradicional tornou-se obsoleta oferecendo menos qualidade nas chamadas, limitações nos serviços que é possível oferecer e um problema logístico para quem tem de as manter. Tudo isto levaria a um lento processo de substituição das redes POTS para as novas redes VoIP.

2.2 Voice over IP e tecnologias associadas

VoIP [3] (Voice over IP) é a transmissão de áudio ou vídeo de uma chamada telefónica através de uma rede que use o protocolo da Internet (IP). Podem ser utilizadas várias tecnologias para alcançar este objectivo como telefones compatíveis com VoIP que são ligados directamente a uma rede IP,

softphones que se podem instalar num computador ou mais recentemente através de browsers que suportem (WebRTC).

VoIP usa codificadores para encapsular e/ou comprimir os dados de áudio captados em pacotes que são depois enviados através da rede IP. Normalmente os pacotes fazem parte de um fluxo do protocolo RTP [4] ou SRTP [5] (Real Time Protocol / Secure Real Time Protocol). O protocolo SIP (Session Initiation Protocol) é frequentemente usado para iniciar, manter e terminar chamadas VoIP. O protocolo SDP (Session Description Protocol) é usado para negociar os codificadores a usar durante a sessão bem como os parâmetros das sessões para assegurar que um receptor é capaz de descodificar o que o emissor transmitiu.

O uso de tecnologias de transmissão com recuso a codificação sobretudo o recurso a uma rede Packet-Switched que permite ter várias chamadas numa única ligação torna o sistema VoIP muito eficiente quando comparado com o seu antecessor analógico que necessitava de ter uma conexão física dedicada á chamada entre os dois terminais.

2.2.1 Protocolos SIP/SDP

O protocolo SIP [6] (Session Initiation Protocol) é um protocolo que para sinalização e controlo de sessões de comunicação em tempo real. Normalmente usado em aplicações VoIP é um protocolo baseado em texto que incorpora muitas características do protocolo HTTP. SIP permite assim criar, modificar e terminar sessões independentemente do protocolo de transporte usado ou do tipo de sessão que está a ser estabelecida.

O protocolo SIP é assim responsável por identificar se um utilizador destino está disposto a realizar as comunicações, por determinar os parâmetros necessários para a sessão, por estabelecer a sessão e finalmente, por gerir a sessão podendo assim terminá-la, transferi-la, modificar parâmetros e chamar serviços.

Normalmente o protocolo SIP é usado em conjunto com o protocolo SDP (Session Description Protocol [7]) que é usado para descrever os parâmetros de uma sessão tais como os codecs a usar para codificar/descodificar os dados, os ritmos binários entre vários outros parâmetros possíveis. Tal como o SIP o protocolo SDP é baseado em texto e pode ser usado de forma independente sem estar integrado no protocolo SIP protocolo.

2.2.2 WebRTC

A tecnologia WebRTC permite a um browser realizar comunicações RTC com qualquer outro dispositivo que o suporte sem o uso de plugin's ou outro qualquer tipo de software adicional. Com esta tecnologia é possível de forma fácil e rápida capturar vídeo e áudio e transmiti-los em tempo real através da internet. O projecto WebRTC [8] consegue criar ligações peer-to-peer entre browsers, para tal o protocolo ICE [9] (Interactive Connectivity Establishment) é utilizado para permitir o estabelecimento da ligação mesmo quando ambos os intervenientes na comunicação estão atrás de uma NAT (Network Address Translation). Assim é apenas necessário que o servidor Web redireccione as mensagens SDP

e os candidatos ICE, para que ambos os intervenientes descubram os IP e porto necessários para enviar as streams de dados de forma directa.

Nos browsers as streams são criadas através da invocação de uma API que controla funcionalidades que estão implementadas nativamente no browser, permitindo o acesso á câmara e ao microfone do utilizador sem a necessidade de instalar software adicional. Esta implementação torna o WebRTC muito fácil de distribuir uma vez que do ponto de vista do utilizador não é necessário realizar nenhum tipo de instalação ou configuração para que tudo funcione. Por motivos de segurança uma boa implementação de WebRTC obriga o utilizador a confirmar o acesso aos dispositivos para cada WebSite que os requisite.

2.2.3 Protocolo ICE

O protocolo ICE, definido pela IETF no RFC 5245 [9], tem como objectivo descobrir o caminho de rede mais adequado para uma dada ligação. Usando esta metodologia é possível estabelecer uma ligação P2P com um dispositivo na mesma rede local ou no outro lado do mundo, através do caminho mais vantajoso para a ligação.

Para que isto seja possível cada um dos intervenientes fica encarregue de descobrir os IP's privados e públicos que lhe estão atribuídos, gerando depois um pacote denominado ICE candidate que é enviado para o peer através de um canal de sinalização que é independente do protocolo. Para descobrir o IP publico o agente que está a gerar os candidatos recorre a um servidor STUN [10] (Session Traversal Utilities for NAT) cuja única funcionalidade, neste caso, é enviar o IP publico para o agente como se pode ver na Figura 3. Esta metodologia é o suficiente para a maioria das NAT's que são utilizadas.

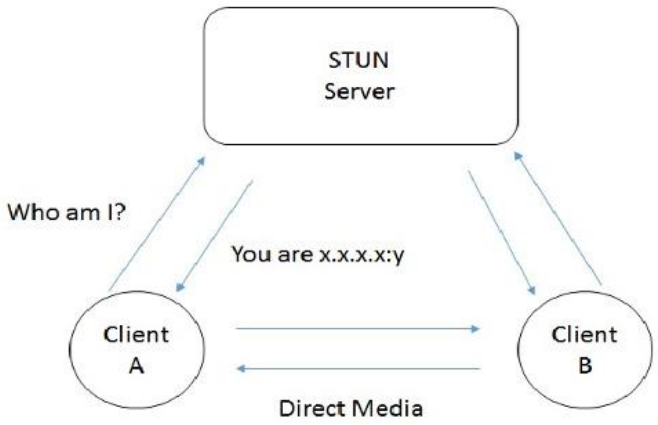

Figura 3 - Servidor STUN [11]

Apesar de o servidor STUN ser uma ajuda externa suficiente para estabelecer a maioria das ligações não é o suficiente para todas. Utilizadores cujos terminais estejam em redes que utilizem NAT's simétricas, ou seja, que trocam o porto e o IP tornam impossível o estabelecimento da sessão usando apenas o servidor STUN. Para resolver este problema é necessário recorrer a um servidor TURN (Traversal Using Relay NAT), ao contrário do servidor STUN o servidor TURN toma parte activa na comunicação reenviando os pacotes para os destinos como se pode ver na Figura 4. Assim a utilização

do servidor TURN é altamente desaconselhada sendo melhor as redes dos terminais usarem uma política de NAT menos agressiva,

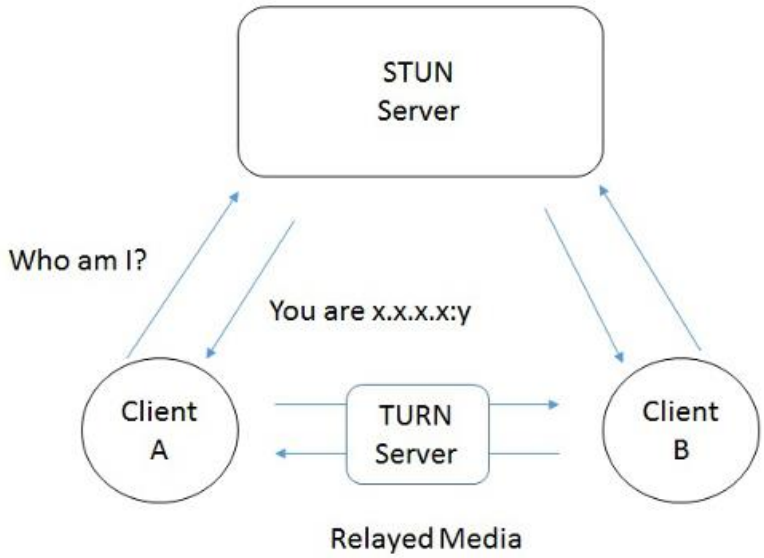

Figura 4 - Servidor TURN [11]

2.2.4 Trickle ICE

O protocolo ICE pode demorar muito tempo a ser executado uma vez que tem de descobrir todos os IP's do cliente sendo necessário realizar várias comunicações através da rede. Isto acaba por atrasar o início das sessões RTP. Para contrariar este problema foi criada uma extensão ao protocolo ICE denominada Trickle ICE [12], esta extensão permite ao agente ICE enviar os candidatos sem ter de esperar por todos, assim na prática o agente ICE vai enviando os candidatos á medida que os gera. O peer que os recebe também pode testar a conectividade e qualidade da conexão á medida que os recebe, assim que tiver um candidato que permita uma ligação com qualidade pode iniciar o envio do seu lado da sessão de imediato. Esta extensão também define que se algum dos restantes candidatos for melhor em termos de desempenho a sessão deve passar a ser feita através do endereço de transporte desse candidato.

Esta extensão torna o protocolo ICE ideal para estabelecer ligações em tempo real uma vez que acelera de forma significativa o processo de estabelecimento de sessão, sendo que nos casos em que os dois intervenientes se encontram na mesma rede privada o processo acaba por ser quase instantâneo uma vez que o primeiro candidato a ser descoberto (IP privado) é o melhor para a sessão.

2.3 Private Branch Exchange

Um Private Branch Exchanger (PBX) é por definição um sistema de comutação de chamadas telefónicas que serve uma organização privada, interligando os terminais da organização, podendo opcionalmente permitir que estes acedam á Public Switched Telephone Network (PSTN) para a realização de chamadas para telefones fora da organização.

Originalmente os PBX eram peças de hardware especializadas que eram usadas em conjunto com as redes telefónicas POTS, mas com o aparecimento da tecnologia VoIP a necessidade de usar hardware especializado desapareceu e os PBX chamados agora de IP-PBX passaram a poder ser servidores a operar dentro de computadores normais reduzindo o custo e dificuldade de implementar um sistema telefónico privado.

Actualmente existem IP-PBX que podem ser usados pelas empresas sem qualquer tipo de custo associado como é o caso do servidor Asterisk [13] e do FreeSWITCH [14]. Estes dois servidores são ambos open-source e de uso livre e gratuito, mas apesar de gratuitos são mantidos por empresas que se dedicam a manter os servidores e a adicionar novas tecnologias á medida que estas vão surgindo. Actualmente tanto servidor Asterisk como o FreeSWITCH suportam tecnologias WebRTC podendo qualquer um destes fazer parte do sistema.

2.4 Servidores Multimédia

O termo servidor multimédia é utilizado para referir qualquer equipamento ou aplicação que se especialize na partilha e/ou entrega de conteúdos multimédia através da rede. Esta definição acaba por abranger uma vasta gama de equipamentos que vai desde grandes servidores que forneçam Vídeo On Demand (VOD) para websites até pequenos servidores domésticos que armazenem ficheiros multimédia e que são vulgarmente designados por Network Atached Storage (NAS).

Neste trabalho o termo servidor multimédia será usado para designar uma aplicação capaz de entregar áudio e vídeo em tempo real, ou seja, usando os protocolos RTP ou SRTP, sendo capaz de comunicar com servidores PBX usando SIP. Um exemplo deste tipo de servidor é o Kurento Media Server [15], que para além de permitir enviar streams RTP/SRTP permite processar vídeo e áudio realizando operações de transcodificação se estas forem necessárias tal como definido no RFC 5567 [16].

2.5 Soluções VoIP/Web existentes no mercado

Existem várias soluções para integrar as novas tecnologias Web com as antigas tecnologias VoIP permitindo ligar os serviços de suporte e vendas das empresas online de forma simples, rápida e escalável.

2.5.1 OnSIP

OnSIP [17] é um serviço pago que oferece uma aplicação Web capaz de substituir totalmente um telefone VoIP permitindo ao seu utilizador ligar para extensões ou pessoas, receber chamadas, gerir múltiplas chamadas e transferir chamadas. O serviço também permite a ligação de uma rede VoIP sendo compatível com dispositivos de diversas marcas.

O serviço permite desenvolver atendedores de chamadas (IVR's), serviços de VoiceMail além disto também permite aos seus utilizadores procurar pessoas através do nome. OnSIP também disponibiliza todos as outras funcionalidades que são esperadas nos telefones VoIP.

O serviço permite ainda uso de software de terceiros para realizar chamadas usando as credenciais de utilizador OnSIP.

2.5.2 Nexmo

O nexmo [18] é outro serviço pago, mas que funciona de forma muito diferente do OnSIP, este serviço apenas disponibiliza API's para construir uma aplicação á medida do cliente. Usando estas API's é possível ligar uma infraestrutura VoIP á cloud podendo fazer e receber chamadas através de um browser. O serviço permite ainda enviar e receber SMS e construir aplicações Web de voz.

O uso das API's é cobrado de forma individual conforme as acções que são feitas muito á semelhança de fornecedor de serviços o cliente é cobrado por chamada e por SMS que faz e no caso da chamada até quando recebe. Se for necessário alugar números de telefone virtuais estes serão cobrados separadamente.

2.5.3 Kurento

O kurento é um servidor multimédia open-source que permite estabelecer ligações através de WebRTC usando com completo suporte do protocolo Trickle ICE. Este servidor multimédia pode também suar as mais tradicionais ligações RTP sem o uso do protocolo ICE e pode ainda receber dados através de uma ligação HTTP para upload de ficheiros. Ao nível interno o Kurento suporta gravação e reprodução de ficheiros de áudio e vídeo, podendo realizar operações de transcodificação se necessário. Além destas funcionalidades padrão de um servidor multimédia o Kurento consegue ainda aplicar filtros às streams que recebe podendo assim adicionar efeitos às imagens.

2.5.4 Bibliotecas WebRTC

Foi possível encontrar duas bibliotecas JavaScript open-source que se dedicam a suportar comunicações WebRTC usando SIP como protocolo de sinalização. Estas bibliotecas permitem assim que um browser possa comunicar de forma directa com um servidor IP-PBX usando SIP, isto torna assim possível integrar uma aplicação Web com a rede telefónica de uma empresa ou organização.

A biblioteca JSSIP [19] permite registar uma aplicação Web através de um WebSocket num IP-PBX usando SIP, além disto automatiza por completo a execução do protocolo ICE encarregando-se de transmitir os ICE candidates de forma automática. Para realizar o registo SIP usando esta biblioteca é necessário passar o endereço do WebSocket ao qual queremos ligar, o URI da extensão com a qual nos queremos registar e a password para essa mesma extensão. Para realizar uma chamada temos que fornecer o URI que queremos contactar e um objecto com opções que permite definir se a chamada é de áudio ou vídeo, e outras opções como o os elementos HTML que irão receber as streams remotas e as locais (caso a aplicação permita mostrar o que está a ser enviado pela camara para o peer).

Outra opção bastante semelhante é a biblioteca SIP.JS [20], esta biblioteca permite realizar as mesmas operações da anterior de uma forma bastante semelhante mas ao contrário da anterior oferece a possibilidade de modificar a forma como uma ligação é estabelecida permitindo que o programador possa optar por outra forma de ligação que não a WebRTC. Isto é muito útil especialmente se quisermos que um servidor NodeJS atenda chamadas. Com esta funcionalidade podemos ter um servidor atender uma oferta SIP e a reenviar o conteúdo SDP para um servidor multimédia, isto permitiria ao servidor NodeJS receber todos os pacotes SIP (o que inclui os DTMF caso estes sejam transmitidos via SIP)

da ligação enquanto que o servidor multimédia lidaria com os pacotes RTP, este tipo de tipologia é interessante para desenvolver um menu IVR em NodeJS.

3 Sistema

O sistema a implementar para resolver o problema não visa substituir as infraestruturas existentes, mas sim estendê-las tentando evitar qualquer tipo de alteração disruptiva. Como se pode ver na Figura 5 o sistema a implementar comunica através de interfaces que já existem no sistema inicial usando o protocolo SIP para comunicar com o PBX e o protocolo HTTPS para interagir com os serviços de autenticação e directório da organização.

Os componentes que farão parte do novo sistema serão o sistema WebRTC que é composto por um servidor Web, um PBX Asterisk e uma base de dados, o servidor IVR que fornecerá menus IVR para todo o sistema e um servidor multimédia que irá gravar mensagens e reproduzir os conteúdos multimédia dos menus IVR.

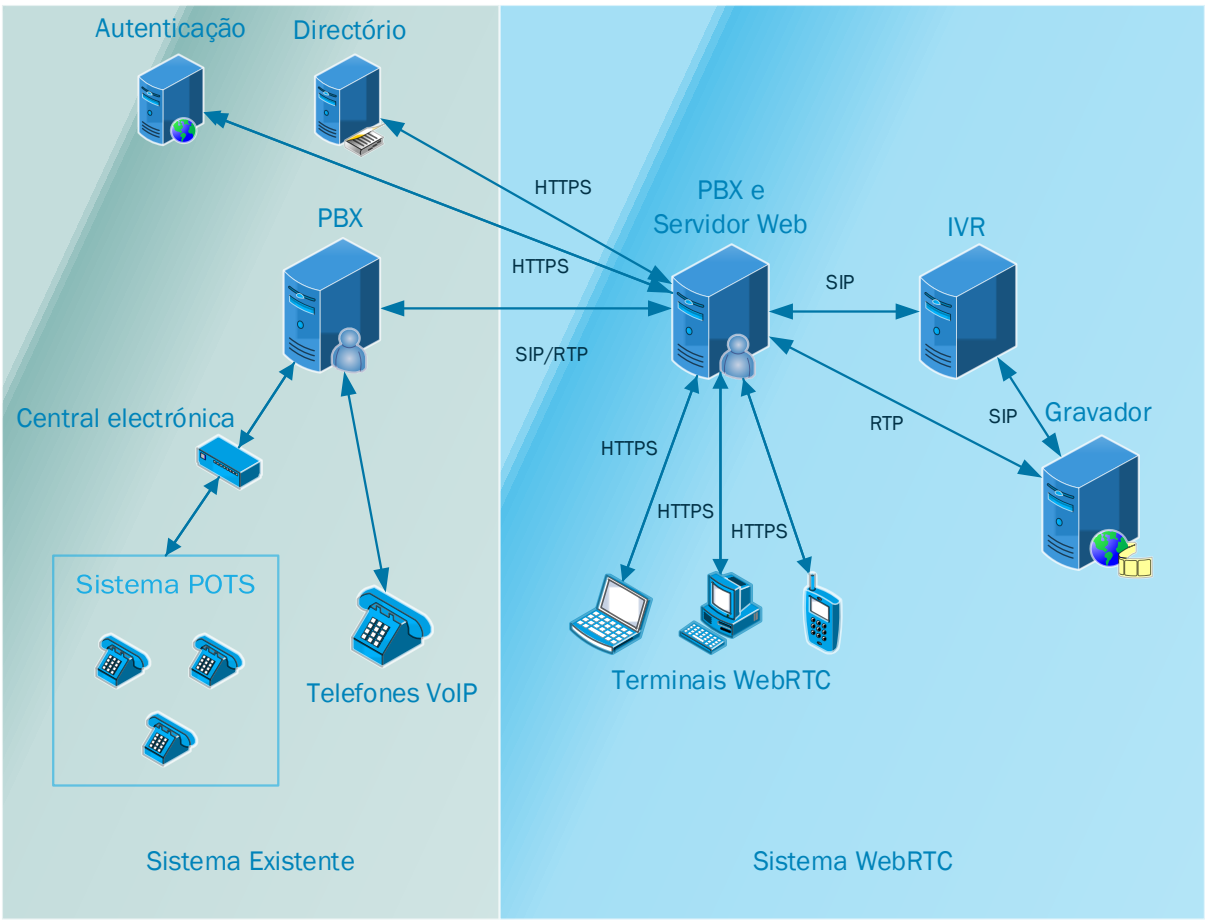

Figura 5 - Arquitectura do geral do sistema

3.1 Requisitos

O sistema deve permitir aos seus utilizadores realizar chamadas entre si através de um browser compatível com WebRTC, o sistema deve autenticar e obter informação sobre os utilizadores usando infra-estrutura existente permitindo assim que novos utilizadores sejam adicionados ao sistema sem necessidade pedir informações ao utilizador. O sistema deve permitir aos utilizadores deixar mensagens de áudio e vídeo a outros utilizadores dando acesso aos mesmos tanto através do browser

como da infra-estrutura VoIP existente. O sistema deve também oferecer a possibilidade deve também oferecer menus IVR usando tecnologias baseadas na Web permitindo assim oferecer serviços aos utilizadores que seriam impossíveis ou muito complicados de realizar usando programas como o Asterisk. Estes menus IVR ao serem escritos em linguagens de alto nível baseadas na Web facilita a gestão do sistema bem como o desenvolvimento de novas funcionalidades.

Usando um browser como cliente o sistema deverá entregar ao utilizador as seguintes funcionalidades:

R 1. Autenticação usando infraestrutura existente;
R 2. Procurar utilizadores pelo nome;
R 3. Comunicar de forma bidirecional independentemente de os terminais usarem tecnologia WebRTC, VoIP ou POTS;
R 4. Garantir a confidencialidade das comunicações feitas através da internet;
R 5. Gravar mensagens de voz/vídeo através de qualquer terminal;
R 6. Agregar as mensagens numa única interface;
R 7. Agregar os vários números de um utilizador;
R 8. Suportar múltiplas sessões de um mesmo utilizador em dispositivos distintos;
R 9. Uma chamada dirigida ao browser de um utilizador deve poder ser atendida em qualquer um dos dispositivos ligados.

3.2 Arquitectura do sistema WebRTC

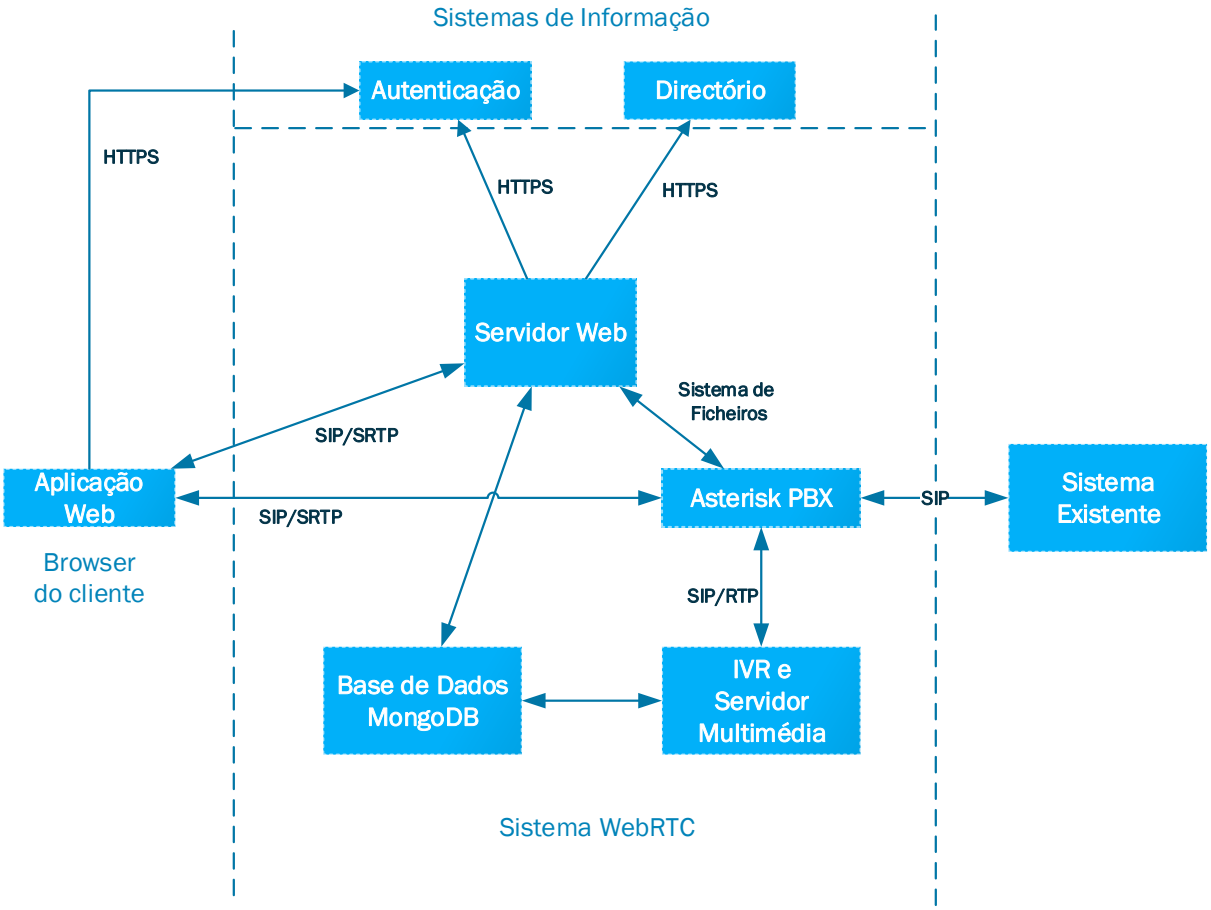

Figura 6 - Componentes do sistema

Os novos componentes do sistema podem ser observados com maior detalhe na Figura 6, na figura podemos observar um Servidor Web, uma base de dados, um PBX mais concretamente um servidor Asterisk, um servidor IVR acoplado a servidor multimédia e por fim do lado do cliente uma aplicação Web.

3.2.1 Servidor Web

O servidor é um componente muito importante do sistema tendo contacto directo com quase todos os outros componentes do sistema sendo mesmo a única excepção o servidor IVR. Esta importância deve-se ao facto do servidor Web executar as seguintes três destines funções:

1. Servir uma aplicação Web que permite aos utilizadores interagir com o sistema;
2. Fornecer um canal de comunicação para a troca de ofertas/respostas SDP e candidatos ICE para permitir aos utilizadores estabelecerem chamadas;
3. Configurar o PBX definindo um utilizador e credenciais que permitam aos utilizadores usar a aplicação para ligar para extensões do sistema VoIP que já existia.

O servidor web tem ainda uma API que permite realizar diversas funções como procurar números de pessoas, fazer chamadas, deixar mensagens para um utilizador, consultar as mensagens recebidas e

adicionar números/extensões ao utilizador. Para implementar estas funcionalidades o servidor necessita comunicar com a base de dados, para armazenar dados referentes aos números dos utilizadores e mensagens enviadas e recebidas. O servidor Web também acede aos serviços de informação da organização para obter números e autenticar cada um dos utilizadores.

3.2.2 Aplicação Web

A aplicação Web fornecida pelo servidor é principal forma de interacção que os utilizadores têm com o sistema permitindo que estes realizem chamadas para outros utilizadores da aplicação, para softphones que estejam associados ao novo sistema ou para extensões do sistema POTS. A aplicação permite ainda que os utilizadores gravem mensagens de áudio e vídeo e as carreguem para o sistema de modo a poderem ser vistas pelo destinatário na sua aplicação web. A aplicação também oferece um sistema básico de procura para permitir aos utilizadores encontrarem os números/extensões de outros utilizadores.

3.2.3 Asterisk PBX

O Asterisk PBX permite aos utilizadores do sistema comunicar com o antigo sistema VoIP. Usando as configurações fornecidas pelo servidor Web o Asterisk permite que as aplicações Web se registem como telefones e que possam assim fazer e receber chamadas de extensões do sistema VoIP. O PBX também interliga o antigo sistema VoIP com o novo sistema IVR permitindo assim que o antigo sistema VoIP possa usar o novo sistema IVR.

3.2.4 IVR e servidor multimédia

O servidor IVR comunica com o Asterisk através de websockets usando o protocolo SIP, para o Asterisk o servidor IVR comporta-se como qualquer outro cliente SIP, registando-se assim que é iniciado e podendo a partir desse momento receber chamadas de voz. O servidor IVR também comunica com o servidor de multimédia (Kurento) controlando-o e permitindo assim que este se possa ligar ao Asterisk para transmitir áudio para os clientes ou gravar o áudio enviado pelos clientes.

3.2.5 Modelos da base de dados

A base de dados do sistema tem 4 tabelas que representam utilizadores, sessões de utilizadores, números dos utilizadores e gravações conforme se pode ver no diagrama da Figura 7.

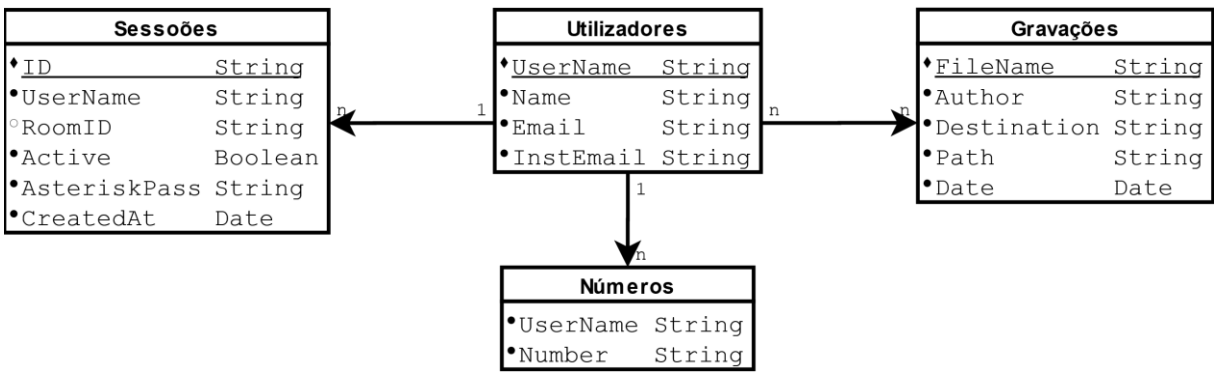

Figura 7 -Diagrama UML da base de dados

A tabela “utilizadores” contém um nome de utilizador como chave primária e é o mesmo que usado para o utilizador realizar o login na aplicação usando o sistema de autenticação da organização. Os restantes da colecção representam informação pessoal do utilizador obtida através da utilização do protocolo OAuth, sendo a única excepção a password do Asterisk que é gerada aleatoriamente, e alterada sempre que o outro utilizador faço um novo login e não exista nenhuma sessão activa.

A tabela “sessões” representa a sessão de um dispositivo do utilizador com o servidor Web e podem por isto existir várias para cada utilizador. Cada registo da sessão contém obrigatoriamente um identificador único que funciona como chave primária, o username do utilizador a que a sessão pertence, um booleano que mostra se a sessão está ativa de momento, ou seja, se está preparada para receber chamadas e um campo com a data em que foi criada. O registo contém ainda um campo chamado RoomID que é null quando a sessão é criada. Este apenas é preenchido após o registo do WebSocket no servidor Web e é actualizado sempre que a aplicação seja carregada, isto porque a acção de recarregar a aplicação leva a um novo registo do WebSocket o que leva á geração de um novo RoomID.

A tabela “números” contém apenas o username a que o número pertence e o próprio número, os números de cada utilizador devem ser obtidos acedendo ao directório da organização através de OAuth para garantir que estes realmente pertencem aos utilizadores. O registo dos números na base de dados de sistema permite que os utilizadores os vejam quando procuram por um utilizador e que possam logo aí fazer uma chamada pelo browser.

Por fim a tabela “gravações” cujos registos são compostos pelo nome do ficheiro (que tem os dados da gravação) no sistema, o autor da gravação ou seja quem o username de quem a gravou, o destinatário ou seja o username do utilizador a que se destina, o caminho para a pasta onde o ficheiro reside e por fim a data em que a gravação foi feita. Cada utilizador pode estar associado a n gravações seja como destinatário ou como autor.

4 Implementação

Esta secção dedica-se a explicar a forma como o sistema da Figura 6 foi implementado explicando com mais detalhe os seguintes componentes:

- Base de dados;
- Servidor Web;
- Aplicação Web;
- Servidor Asterisk;
- Sistema IVR.

No fim desta secção encontra-se ainda uma explicação detalhada da implementação das principais funcionalidades do sistema, analisando a forma como os vários componentes interagem e as mensagens que trocam passo a passo.

4.1 Base de dados

A base de dados de dados escolhida para o sistema foi o MongoDB, a escolha de uma base de dados NoSQL deve-se ao facto deste sistema não necessitar das funcionalidades oferecidas por bases de dados SQL como as transacções e facilidade de executar queries complexas a várias tabelas em simultâneo. Assim sendo uma base de dados mais simples e menos estruturada com menor rigidez permite que o modelo de dados seja implementado com maior facilidade, além disto a base de dados também se torna mais escalável bastando apenas acrescentar mais máquinas á base de dados para melhorar o desempenho.

4.2 Servidor Web

O servidor web tem como função disponibilizar uma API que permita á aplicação Web configurar-se e realizar chamadas WebRTC e fazer upload de gravações, configurar o PBX Asterisk adicionando os utilizadores aos ficheiros de configuração e servir os ficheiros que compõem a aplicação Web.

A linguagem de programação que foi utilizada para a criação do servidor foi NodeJS, esta escolha deveu-se ao facto de ser muito simples criar servidores Web com ferramentas muito poderosas como ExpressJS. Para além dos pacotes disponíveis que facilitam o processo o facto de usar JavaScript praticamente igual ao que se encontra no desenvolvimento no browser facilita a tarefa de desenvolver ambos uma vez que a sintaxe é a mesma.

O servidor usa os seguintes pacotes NodeJS:

- Express;
- SocketIO;
- Cookie-Session;
- Body-Parser;
- Multer;
- Simple-Oauth2;

- Random-ID;
- Mongoose;
- Memory-Cache.

O pacote ExpressJS é o pacote que gere o servidor Web sendo responsável pelo processamento de pedidos HTTP e pela definição de rotas e pré processamento dos pedidos HTTP. O SocketIO é o WebSocket que foi escolhido para este projecto, ele usa o mesmo agente HTTPS que o Express e por isso existe no mesmo porto.

Os pacotes Cookie-Session, Body-Parser e Multer estendem as funcionalidades nativas do Express. O Cookie-Session configura um cookie para usar nos browsers que o suportem, permitindo armazenar dados desde que estes não excedam o tamanho máximo suportado para os cookies, também assina os cookies de modo a que as informações não possam ser forjadas. O Body-Parser faz o pré-processamento da informação recebida nas mensagens, por exemplo se o texto enviado é JSON este pacote analisa o conteúdo e coloca o objecto adequado para uso futuro na rota.

O pacote Multer é um pacote parecido com o Body-Parser mas especializado no upload de ficheiros, assim sempre que é recebido conteúdo do tipo *multipart/form-data* o Multer guarda o ficheiro na pasta destinada aos uploads.

Simple-Oauth2 é um pacote com uma API que simplifica a utilização do framework OAuth 2.0 [21]. Este pacote é necessário para usar login do sistema fénix como método de autenticação de um utilizador bem como para obter dados pessoais básicos como o nome e o e-mail para uso no servidor.

O pacote Random-ID é usado para gerar indentificadores aleatórios que são usados para identificar as sessões.

Mongoose é um pacote que disponibiliza uma API para aceder a uma base de dados MongoDB onde são guardados os dados do servidor referentes aos utilizadores.

Por fim a Memory-Cache permite através de um esquema de chave-valor guardar dados de forma temporária, este pacote é usado para armazenar dados de chamadas que estão pendentes.

4.2.1 API Web

A comunicação entre a aplicação Web desenvolvida e o servidor Web é feita através de uma API que se divide em duas componentes, uma componente HTTP que foi desenvolvida seguindo os princípios que regem as API's REST e uma outra componente WebSocket que é usada para a sinalização das chamadas WebRTC.

A Tabela 1 mostra as rotas da API web indicando o path do URL e dando uma breve descrição da função de cada rota Web.

Tabela 1 – Lista das rotas com método e descrição resumida do funcionamento.

Rota	Descrição
GET / (raiz)	Obtém o ficheiro layout.html que contém toda a informação necessária para iniciar a WebApp.
GET /login	Inicia o processo de login redireccionando o utilizador para a página de LogIn do sistema fénix.
GET /auth	Ponto de destino após o login no sistema fénix estar concluído, realiza todas as operações necessárias para concluir o processo de LogIn, obtendo os dados necessários no sistema fénix, criando a sessão para o browser e configurando, se necessário, o Asterisk. Após iniciado redireciona o cliente para a raiz para que a WebApp reinicie.
GET /logout	Destrói a sessão que está na base de dados e o cookie de modo a terminar a sessão.
GET /v1/person	Obtém os dados pessoais do utilizador que faz o pedido. O servidor responde com 200 “Ok” as informações do utilizador são enviadas no corpo da mensagem.
GET /v1/person/:username	Obtém os dados do utilizador identificado pelo username. O servidor responde com 200 “Ok” as informações do utilizador são enviadas no corpo da mensagem.
GET /v1/person/:username/online	Questiona o servidor sobre se um dado utilizador está online ou offline. Em caso de sucesso o servidor responde com 200 “Ok” e envia no corpo da mensagem *true* se estiver online e *false* se está offline.
POST /v1/callrequest	Inicia o processo de chamada. Deve ser colocado em formato JSON uma mensagem no formato {destUserName: String}. Após receber um POST válido o servidor contacta o cliente alvo da chamada, e responde ao cliente como sucesso do progresso da chamada. O servidor responde com 201 “Created”
DELETE /v1/callrequest/:callRequestID	Esta rota é destinada para o envio de uma resposta a um pedido de chamada enviado pelo servidor ao cliente. O servidor responde com 200 “Ok” e envia no corpo da mensagem o identificador da mensagem caso o utilizador tenha escolhido atender a chamada caso contrário responde com 204 “No Content”.
POST /v1/record	Esta rota serve para o upload de um ficheiro de vídeo ou áudio contendo uma gravação. Para esta rota deve ser enviado no corpo um objecto FormData, com os parâmetros blob e destenation sendo estes os dados do ficheiro e o utilizador para quem se quer

	enviar a mensagem respectivamente. O servidor responde com 201 “Created” em caso de sucesso.
GET /v1/record/:filename	Faz o download de uma gravação identificada pelo nome do ficheiro no URL, o servidor só permite esta operação caso o utilizador seja o destinatário da mensagem. Em caso de sucesso o servidor responde com 200 “Ok” e inicia do envio do ficheiro.
DELETE /v1/record/:filename	Apaga uma mensagem identificada através do identificador passado através do URL, o servidor só permite esta operação caso o utilizador seja o destinatário ou autor da mensagem. Em caso de sucesso o servidor responde com 204 “No Content”
GET /v1/person/:username/record	Obtém os meta-dados das mensagens recebidas pelo utilizador. Em caso de sucesso o servidor responde com 200 “Ok” enviando no corpo da mensagem uma lista com os dados das mensagens/gravações.
POST /v1/number	Adiciona um número á base de dados. No corpo da mensagem deve estar um parâmetro number contendo o número que se pretende adicionar. O servidor responde com 201 “Created” em caso de sucesso.
GET /v1/number/:username	Obtém todos os números associados a um nome de utilizador passado através do URL. O servidor responde com 200 “Ok” em caso de sucesso.
DELETE /v1/number/:number	Apaga um número passado através do URL e que esteja associado ao utilizador. O servidor responde com mensagem 204 “No Content” em caso de sucesso.
GET /v1/asterisk/:username	Obtém a password do Asterisk para a o utilizador fornecido. Em caso de sucesso o servidor responde com 200 “Ok”

A Tabela 2 mostra as mensagens que o WebSocket do servidor pode receber vindas da aplicação Web dando uma breve descrição sobre o funcionamento de cada uma.

Tabela 2 - Lista das mensagens que o WebSocket pode receber

Mensagem	Descrição
Register	Esta rota serve para registar um cliente sockteIO na aplicação, após a conclusão bem-sucedida o cliente fica associado a uma sessão na base de dados. O registo recebe em formato JSON uma mensagem com a seguinte estrutura {id: String, username: String}. Sendo o id a string que foi gerada durante o processo de autenticação e o username o nome de utilizador do sistema fénix. Após a conclusão bem-sucedida o servidor responde com uma mensagem */registred* caso contrário responde com ***/refused***.

ICECandidate	Rota que faz o redireccionamento dos candidatos ICE para os clientes que estão na sala indicada pela variável dest, devendo a mensagem ter a seguinte estrutura {dest: destino, candidate: ICEcandidate}. O destino é o indentificador/sala fornecido no início da chamada.
hangUp	Rota serve para sinalizar o fim de uma chamada ao par. A mensagem deve incluir o par destino. O corpo da mensagem deve apenas ter o ID do destino que deve ser identificador/sala fornecido no inicio da chamada.
leaveroom	Rota que informa o servidor que um cliente pretende deixar uma sala. A sala corresponde ao identificador/sala fornecido no inicio da chamada. Este comando é necessário para terminar de forma permanente uma chamada.
setUpMessage	Rota para redireccionar uma mensagem de um cliente para outro. A mensagem deve ter a seguinte estrutura, {dest: destino, message: mensagem}. O destino é o identificador/sala fornecido no inicio da chamada.
disconnect	Esta rota é activada automaticamente sempre que um cliente se desliga. Quando esta rota é activada o servidor muda o estado da sessão que corresponde ao cliente para inactivo.

A Tabela 3 mostra as mensagens que o servidor Web pode enviar para o WebSocket da aplicação dando também uma breve descrição sobre o significado e função de cada uma.

Tabela 3 - Lista das mensagens que o WebSocket pode enviar

Mensagem	Descrição
registered	Mensagem enviada pelo servidor quando o registo é bem-sucedido. Se o servidor não enviar esta resposta significa a falha no registo.
call	Mensagem enviada pelo servidor quando um utilizador quer iniciar uma chamada telefónica. Esta mensagem tem a seguinte estrutura {name: nome, callRequestID: ID}. O campo name contém o nome completo do utilizador que iniciou a chamada, o callRequestID tem o identificador que é necessário para identificar o pedido de chamada no servidor devendo ser enviado de volta na resposta. Nesta fase do processo o nome do utilizador já foi autenticado pelo servidor.
setupmessage	Mensagem contendo uma descrição SDP gerada pela API PeerConnection do projecto WebRTC, esta mensagem não contém nenhuma informação adicional especifica deste sistema, a mensagem pode ser uma oferta ou uma resposta tendo a aplicação Web de distinguir as duas através do atributo tipo. O servidor só envia

	esta mensagem após confirmar que o gerador da mensagem está numa chamada com o destinatário.
ICECandidate	Esta mensagem contém um candidato ICE gerado pela API do projecto WebRTC devendo ser adicionado á ligação que está a ser estabelecida. O correcto reencaminhamento destas mensagens assegura o bom funcionamento do protocolo ICE e da sua extensão Trickle ICE.
disconnect	Esta mensagem é recebida sempre que existe uma perda de contacto com o servidor.
callrefused	Enviado pelo servidor quando uma chamada é recusada pelo destinatário.
callexpired	Enviado pelo servidor quando passado um certo período o destinatário não respondeu.
callReady	Enviado pelo servidor quando processo de sinalização para a chamada pode ser iniciado. O corpo da mensagem contém a sala/identificador do SocketIO a ser usada para a sinalização.

4.3 Aplicação Web

A aplicação Web foi escrita em JavaScript utilizando algumas funcionalidades do ECMAScript 6 e é executada em “strict mode” para permitir um processo de DEBUG mais simples e organizado.

A aplicação necessita de se manter em constante execução desde que é iniciada pelo que o carregamento de código HTML deve ser feito utilizando JavaScript. Para facilitar esta tarefa usou-se o framework AngularJS.

Os pacotes Angular e Angular-Route são usados para permitir criar de forma simples uma aplicação com várias páginas Web sem que o browser interrompa o código que está a ser executado em segundo plano. O pacote Angular-Route permite que, quando o URL muda, a aplicação substitua partes da página Web carregando um novo código HTML que substitui o antigo que está dentro da divisão marcada com a tag “ng-view”. Associado a cada URL está um controlador que contém o código JavaScript a executar nessa página o que ajuda a melhor organizar o código. É de notar que a mudança no URL não acciona uma actualização da página apenas parte dela é actualizada pelo que os serviços que estão em execução em segundo plano (ligação do WebSocket ao servidor web e ao Asterisk) não são interrompidos.

O AngularJS permitiu ainda dividir as tarefas da aplicação em diversos serviços divididos por diversos ficheiros. Esta possibilidade tornou o código mais simples aumentando de forma significativa expansibilidade e tornando a manutenção mais simples.

Para criar a aplicação foram ainda necessários outros pacotes:

- JQuery
- JSSIP
- SocketIO

- Adapter.js

O pacote JQuery é utilizado pelo AngularJS, o carregamento desta biblioteca é assim necessário para assegurar o correcto funcionamento do Angular.

A biblioteca JSSIP é necessária para implementar um telefone SIP no browser, não sendo a única opção disponível para realizar esta tarefa é uma das mais estáveis e que oferece mais flexibilidade. Usando JSSIP a aplicação pode registar-se no servidor Asterisk para fazer e receber chamadas.

O pacote SocketIO é o WebSocket da aplicação, é usado para receber comunicações vindas do servidor permitindo que que o servidor possa contactar o cliente a qualquer altura. Também permite através de um "heart beat" integrado que o servidor tome conhecimento quando o cliente é encerrado de forma inesperada.

O Adapter.js permite que uma aplicação WebRTC funcione em browsers com diferentes implementações do projecto WEbRTC. Usando este pacote é possível desenvolver a aplicação usando uma API generalizada deixando para esta biblioteca a compatibilidade com as camadas de software de baixo nível permitindo assim compatibilidade com várias versões do projecto WebRTC e também com browsers diferentes.

4.3.1 Organização do código

Como já foi referido a aplicação usou o AngularJS como framework o que permitiu separar o código da aplicação pelos seguintes serviços:

- Dados pessoais – Guarda dados pessoais tal como o nome e IstID e palavra-chave dos Asterisk.
- WebSocket – Define a forma como a aplicação lida com as mensagens vindas do servidor através do protocolo WSS usando a versão cliente do SocketIO. Também é responsável por registar o cliente no servidor.
- Chamada SIP – Agrega todo o código que constrói o telefone SIP da aplicação com a biblioteca JSSIP no centro.
- Chamada para o Browser – Agrega o código que permite realizar chamadas directas de browser para browser.
- Gravar mensagem – Agrega o código que permite gravar uma mensagem e posteriormente carregá-la para o servidor.

Estes serviços funcionam de forma independente tendo apenas algumas dependências com o WebSocket por o utilizarem como meio de comunicação com o servidor.

Estes serviços são usados pelos controladores das páginas permitindo que estes contenham apenas o código que controla o conteúdo das páginas. Este esquema torna as interacções da aplicação com os servidores independentes da geração das páginas Web.

4.4 Servidor Asterisk

A versão do servidor Asterisk [13] usada neste sistema foi a 13 sendo esta a versão mais recente, com suporte de longo prazo. Esta versão foi compilada a partir do código fonte usando o channel driver PJSIP. Para esta versão também foi necessário instalar o codec de áudio Opus.

4.4.1 Configuração dos módulos do Asterisk

Será necessário configurar diversos módulos de modo a permitir que o servidor Asterisk tenha o comportamento desejado. Os módulos são os seguintes:

- HTTP – Módulo que controla o servidor Web que se encontra integrado no servidor, este modulo é necessário para que o servidor possa receber pedidos vindos de browsers através de WebSockets;
- RTP – Controla as opções que dizem respeito aos fluxos RTP como os portos que podem ser usados e os protocolos usados para o estabelecimento de uma sessão RTP;
- DialPlan – Este módulo é o responsável por definir as extensões decidindo o destino dos pedidos de chamada que alcançam o servidor. Este módulo define vários contextos, mas apenas os que são permitidos no módulo PJSIP para o utilizador podem ser usados para tratar a chamada;
- PJSIP – O Channel Driver que contém as definições dos utilizadores e das camadas de transporte bem como as credenciais de autenticação. Este módulo é o responsável por permitir ou bloquear conexões chegam ao servidor bem como seguir e contactar utilizadores que estejam registados no servidor.

Para utilizar WebRTC é necessário que o servidor Web use WSS e para isto são necessários um certificado e uma chave RSA, o ficheiro contendo a chave deve ser especificado no ficheiro de configuração bem como o endereço IP e porto onde se pretende receber as ligações. Um exemplo de configuração pode ser encontrado no anexo A.

No módulo RTP é necessário adicionar várias opções que ajudam a garantir que o servidor Asterisk é capaz de estabelecer ligações mesmo através de NAT's, para atingir este objectivo é necessário activar o protocolo ICE e definir um servidor STUN. Para além disto é necessário instruir o Asterisk para usar o mesmo porto para enviar e receber dados das streams RTP *rtp_symetric=yes.* A configuração do modulo RTP pode ser vista no anexo C.

O dialpan é simples de configurar contendo apenas uma lista com as extensões configuradas para cada contexto, para que o sistema funcione foi apenas necessário um modelo/patern de extensão que permite ao Asterisk definir uma extensão para todos os utilizadores usando o username que é único no sistema como extensão. A função *PJSIP_DIAL_CONTACTS* foi usada no Dial para que o Asterisk contacte todos os URI's do utilizador destino que se encontram registados no Asterisk.

Este ficheiro também define uma Macro que envia o número para o qual um utilizador tenta ligar como DTMF's para o servidor IVR que lida com chamadas não atendidas. A configuração completa pode ser encontrada no Anexo D.

4.4.2 Configuração do Channel Driver PJSIP

O modulo PJSIP é o mais complexo e o que requer mais configuração uma vez que é responsável pelo registo de utilizadores, da aplicação Web e das extensões telefónicas, e estabelecimento de chamadas. Aqui são definidos os utilizadores e descritos os codecs que cada um pode suportar bem como os protocolos de transporte que podem ser usados nas chamadas. As credenciais de autenticação também são definidas neste modulo. A Figura 8 mostra todos os objectos de configuração que existem bem como a relação entre os mesmos.

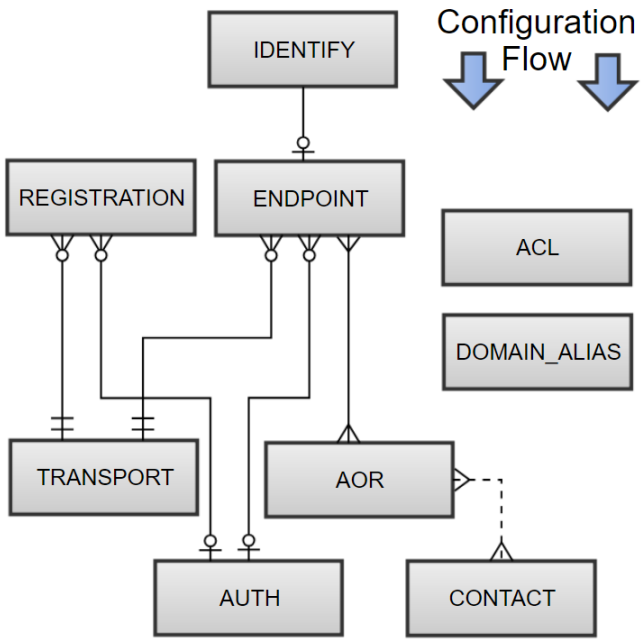

Figura 8 - Relações entre os objectos de configuração PJSIP [22]

De forma resumida a função de cada um destes objectos pode ser descrita da seguinte forma.

- **Transport** – Representa as opções da camada de transporte definindo as interfaces e porto a usar pelo servidor para receber pedidos bem como o protocolo. No caso de se usar encriptação também é aqui que se devem especificar os certificados e as chaves de segurança.
- **Auth** – Este objecto representa credenciais de autenticação sendo geralmente usado para definir um par utilizador/chave. A chave pode ser escrita em texto simples no ficheiro de configuração ou na forma de uma hash MD5.
- **AOR** – AOR ou Address-of-Record é um URI que aponta para um domínio com um serviço de localização que mapeia o URI para outro onde o utilizador pode estar disponível segundo a definição do RFC 3261 [6] . No Asterisk um AOR acaba por se traduzir numa lista de URI's onde um utilizador está disponível, um novo registo é adicionado sempre que o utilizador se regista e apagado sempre que o Asterisk verifica que ele já não está disponível no URI do registo.
- **Endpoint** – Representa o objecto criado quando uma chamada se inicia e descreve os codecs que podem ser usados, se a sessão RTP deve ser cifrada. Se for cifrada também contém as chaves que devem ser usadas bem como o protocolo, os parâmetros a usar para estabelecer

a sessão também são descritos nesta secção. como o uso do protocolo ICE ou de rtcp_mux (que mistura os pacotes de controlo com os pacotes de dados na mesma num só fluxo). Esta secção também define o contexto para o qual a chamada deve ser encaminhada no Dialplan do Asterisk. Os objectos AOR e Auth a usar também são definidos nesta secção sobre a forma de uma referência.

- **Register** – Esta secção permite que um servidor Asterisk comunique com outro usando um endpoint definido para o efeito, isto permite ao Asterisk reencaminhar chamadas que tenham como destino a redes VoIP.

Para que o WebRTC funcione é necessário que o servidor Asterisk suporte WSS (Web Socket Secure). Para além das definições do servidor HTTP definidas anteriormente é necessário criar o objecto e indicar o IP e porto no qual se pretende receber os pedidos SIP.

Para receber pedidos de telefones SIP que não suportem cifra é necessário configurar um modo de transporte que use o protocolo UDP. Para isto é apenas necessário definir o IP e porto que irão receber os pedidos SIP.

Finalmente para utilizadores que usem telefones SIP que suportem cifra é necessário definir um modo de transporte TLS (Transport Layer Security) que permita garantir a confidencialidade das comunicações desde a primeira etapa da sinalização SIP. Para esta configuração além de ser necessário definir o IP e porto é preciso também fornecer uma chave RSA e um certificado para o servidor usar para encriptar os dados.

Os objectos Auth em PJSIP são simples definindo apenas um par nome de utilizador password ou se for pretendido uma hash da password. Neste sistema cada utilizador deve ter a sua password pessoal, esta password é gerada pelo servidor Web quando um novo utilizador utiliza o sistema pela primeira vez. A chave é escrita pelo próprio servidor nos ficheiros de configuração do Asterisk sendo depois utilizada a CLI (Command Line Interface) para recarregar o ficheiro de configuração.

O Asterisk gere os AOR's de forma automática sendo apenas necessário definir o número máximo de registos que pretendemos permitir a um só utilizador, este número na prática traduz-se no número máximo de dispositivos que o utilizador pode ter ligados ao servidor Asterisk em simultâneo.

Os Endpoints têm de ser configurados de modo a que seja possível suportar chamadas WebRTC, para isto é necessário activar o uso de DTLS e fornecer a chave e certificado a usar. O Endpoint deve permitir chamadas com origem no servidor Asterisk e no cliente pelo que o ***dtls_setup*** deve ser definido como ***actpass***. O parâmetro ***dtls_verify*** deve ser definido como ***fingerprint*** , este parâmetro controla a forma como uma stream SRTP é associada ao canal de sinalização. O uso do protocolo ICE do perfil AVPF e da multiplexagem dos pacotes RTCP também deve ser activado para permitir estabelecer ligações através de NAT's. A utilização do codec de áudio Opus deve também ser activada.

4.5 Sistema IVR baseado em NodeJS

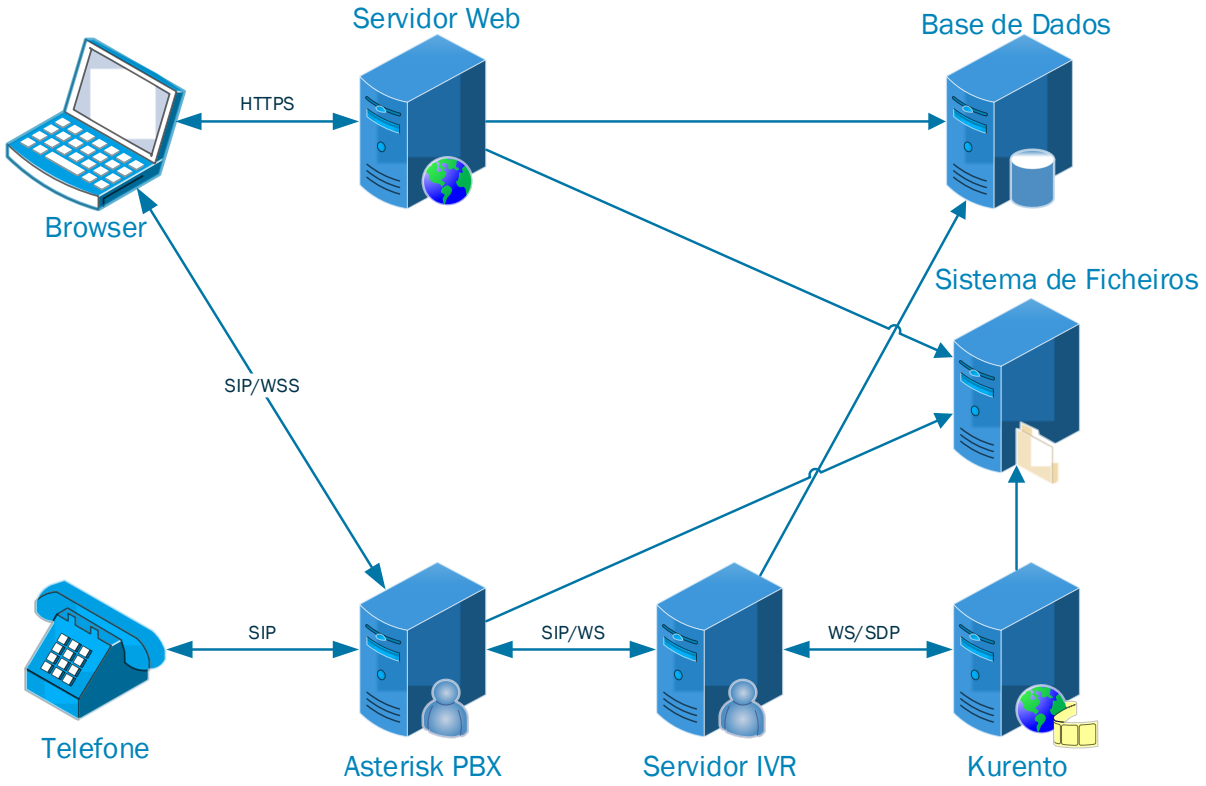

Figura 9 - Arquitectura geral do sistema IVR

O servidor Asterisk tem a capacidade de proporcionar menus IVR para os clientes podendo gravar mensagens voz, criar menus complexos para apoio redirecionar chamas e se for necessário pode manter contexto entre chamadas. Infelizmente o suporte do Asterisk para aceso a bases de dados é bastante limitado permitindo apenas interagir com bases de dados SQL e de forma muito limitada. Este facto torna o Asterisk um servidor inviável quando queremos integrar serviços Web que utilizem bases de dados NoSQL. Para além desta limitação também é impossível invocar API's web durante a execução do menu IVR, o que limita o dinamismo que o menu pode ter.

Assim foi necessário adicionar um componente ao sistema que permitisse criar um sistema IVR flexível que permita dar possibilidade de interagir com uma base de dados ou qualquer outro serviço web de forma simples, dando assim a possibilidade de integrar o sistema VoIP com aplicações existentes. O resultado foi um sistema igual ao descrito na Figura 9.

4.5.1 Tecnologias utilizadas

O servidor multimédia utilizado foi o Kurento tendo sido seleccionado por ser compatível com o Asterisk e com WebRTC além disto é capaz de gravar chamadas e de reproduzir conteúdos multimédia.

A linguagem de programação usada foi JAvaScript usando NodeJS mais uma vez por ser simples criar um servidor que se registe no servidor Asterisk através de um WebSocket e por ser uma das linguagens suportadas para interagir com o servidor multimédia existindo uma biblioteca que assiste na conexão e controlo do servidor.

O servidor usa a biblioteca SIP.JS [20] para lidar com as mensagens SIP, esta biblioteca foi escolhida por permitir que seja executada fora do ambiente do browser, utilizando o WebSocket padrão do NodeJS, e por permitir que o código que gera as respostas aos pedidos de chamada que chegam seja completamente substituído por outro através da injecção de uma fábrica durante a instanciação da biblioteca. Esta funcionalidade é vital uma vez que é necessário que a oferta SDP que é enviada pelo Asterisk quando a chamada é atendida seja enviada para o servidor multimédia para que este gere a resposta. Por padrão estes tipos de bibliotecas tentam usar o modulo WebRTC que costuma estar presente nos browsers ou aplicações para gerar uma resposta, o que não é aplicável neste caso.

4.5.2 Servidor IVR

O servidor IVR tem como função atender chamadas originadas pelos utilizadores e reproduzir mensagens gravadas através da manipulação do servidor multimédia. Para isto é necessário que o servidor use a biblioteca SIP.JS para se registar no Asterisk, neste registo o servidor IVR comporta-se como o telefone SIP normal fornecendo os seus dados de autenticação e registando-se.

O servidor IVR não lida directamente com as streams RTP ele limita-se apenas a receber mensagens SIP originadas no cliente e a agir em conformidade com o menu que está implementado (Figura 10). Assim o servidor pode receber os eventos DTMF como mensagens SIP mudar os sons que são reproduzidos pelo servidor multimédia que esse sim recebe e uma streams de pacotes RTP.

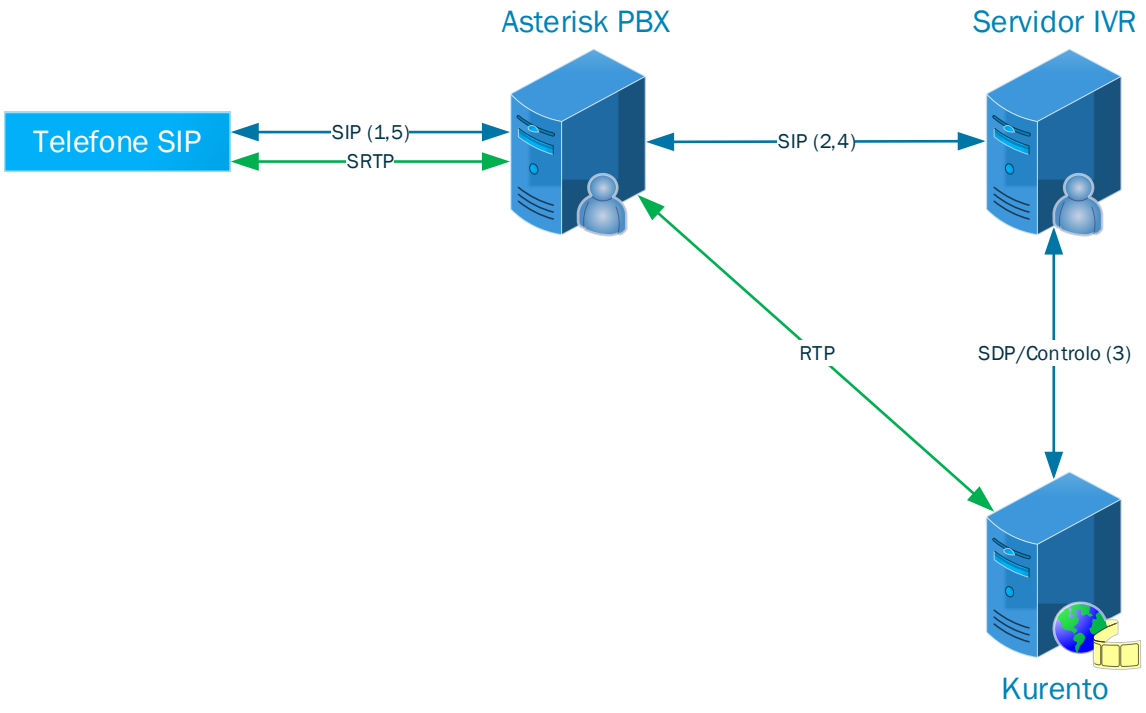

Figura 10 - Fluxos RTP numa chamada envolvendo o servidor IVR

O servidor IVR foi implementado de forma a poder ter vários menus no mesmo servidor sendo que cada menu se regista com uma extensão diferente no servidor Asterisk permitindo assim aceder a um ou a outro através do número da extensão.

Um menu IVR funcional é constituído pelos seguintes módulos:

- ua (User Agent) – Este modulo define o que faz o registo o servidor Asterisk e define como devem ser processadas as mensagens SIP recebidas. Este modulo é construído á volta do pacote SIP.JS.
- menu – Modulo que define o comportamento que se pretende que um menu tenha, este modulo é instanciado pelo modulo UA após um registo bem-sucedido no servidor Asterisk.
- kurento-client – Módulo fornecido pelos programadores do servidor Kurento que permite controlar o servidor multimédia. Este modulo cria um objecto chamado **MediaPipeline** que é usado para gerar uma resposta SDP e mais tarde, já no menu, é utilizado para reproduzir e/ou gravar conteúdos.
- db – É o mesmo modulo que foi utilizado no servidor Web para realizar transacções com a base de dados aqui é usado para guardar os dados de novas gravações que são criadas fazer playback das existentes.

4.5.2.1 Registo do servidor IVR no Asterisk

O registo no servidor Asterisk começa quando o servidor IVR é iniciado enviando uma mensagem SIP Register para o Asterisk. Esta mensagem contém as informações de autenticação que correspondem ao endpoint de cliente que foi configurado no servidor Asterisk. O servidor Asterisk recebe a mensagem e após conferir as credencias responde com um SIP 200 OK. Depois deste registo o servidor IVR pode receber chamadas vindas do servidor Asterisk.

4.5.2.2 Chamada entre um cliente e o sistema IVR

O estabelecimento de uma chamada entre um cliente o servidor IVR segue os seguintes cinco passos visíveis na Figura 10.

1. Chamada é iniciada com o cliente a usar um telefone SIP para contactar com o servidor Asterisk para ligar a uma extensão que pertence ao sistema IVR.
2. O Asterisk reencaminha o pedido para a instância do servidor IVR á qual a extensão pertence enviando também uma oferta SDP.
3. Ao receber a chamada o servidor IVR começa por criar um novo pipeline no servidor Kurento e de seguida adiciona um endpoint RTP ao pipeline do Kurento, este endpoint irá permitir que o Kurento envie e receba streams RTP. Para que o recém-criado endpoint RTP possa enviar dados para o Asterisk o servidor IVR envia a oferta SDP que recebeu do Asterisk. O servidor Kurento gera uma resposta SDP depois de processar a oferta enviando-a para o servidor IVR.
4. Depois de receber a resposta SDP do Kurento o servidor IVR envia a mesma para o Asterisk permitindo que a comunicação bidirecional entre os dois comece.
5. O Asterisk pode nesta fase iniciar uma chamada separada com o cliente e reenviar os dados que recebe do kurento para o cliente e vice-versa. Outra hipótese é tentar enviar a resposta SDP que recebeu do Kurento para o cliente de modo a que estes estabeleçam uma ligação directa. Embora o segundo cenário seja o mais desejável é muito difícil garantir a compatibilidade entre o servidor Kurento e o cliente que pode estar a usar WebRTC, um

qualquer softphone ou até um telefone VoIP tradicional, esta dificuldade torna a primeira solução muito mais robusta uma vez que funciona sempre que o Asterisk seja compatível com o dispositivo do cliente.

4.5.2.3 Funcionamento dos menus

Depois do estabelecimento de uma chamada o menu é iniciado, para que isto seja possível o menu tem de ter uma função chamada init que deve definir o que acontece quando o utilizador envia um sinal DTMF e o som de apresentação padrão do menu. Para que isto seja possível é passado para o menu a sessão SIP.JS que foi gerada na inicialização da chamada em conjunto com o pipeline que foi criado para controlar o servidor Kurento.

Os menus devem ser capazes de orientar várias chamadas em simultâneo pelo que não pode se pode usar variáveis do menu para guardar informação sobre o estado das chamadas. O estado deve ser mantido no objecto Session, este objecto é ideal uma vez que é automaticamente gerado no início da chamada e possui um campo camado ***data*** que deve que existe especificamente para acomodar dados específicos de cada implementação não sendo por isso modificado de nenhuma forma pelo normal funcionamento do objecto Session. Para além disto este objecto é passado para todos os handlers dos eventos que ocorrem na chamada incluído, obviamente, quando um DTMF é recebido. Assim sendo o objecto Session no seu campo data deve conter referências para o pipeline do Kurento e para todos os objectos que estão no pipeline a cada dado momento de modo a que possam ser alterados ou destruídos no futuro.

Para ser possível a um menu IVR ter vários submenus a função que lida com os eventos DTMF é definida de modo a chamar uma função que cuja referência está guardada numa variável da sessão. Este esquema permite que a variável seja definida com a referência para o submenu activo a no momento podendo ser alterada a qualquer momento para outro valor passando assim a nova função a ser executada. Assim é possível trocar de submenu sem mexer no handler dos eventos DTMF que é passado para o modulo SIP.JS.

O Kurento é capaz de realizar reprodução de mensagens em cascata usando um PlayerEndpoint, este objecto informa permite definir um evento para quando o ficheiro que está a ser reproduzido termina permitindo assim destruir o PlayerEndpoint actual e criar outro com o mesmo som ou, se for necessário, repetir o mesmo ficheiro de som usando o mesmo PlayerEndpoint. O servidor é capaz de criar novos Endpoints com um atraso imperceptível para o utilizador isto deve-se ao facto da chamada não ser alterada de nenhuma forma, a única coisa que é alterada é a fonte do som pelo que o atraso é composto apenas pelas seguintes operações(Figura 11):

- Envio de ordens do Servidor IVR para o Servidor Kurento
- Processamento das ordens recebidas do servidor IVR
- Acesso ao novo ficheiro a transmitir (Tempo de acesso ao disco, se necessário)
- Inicio da transmissão do ficheiro (Pode envolver transcodificação)

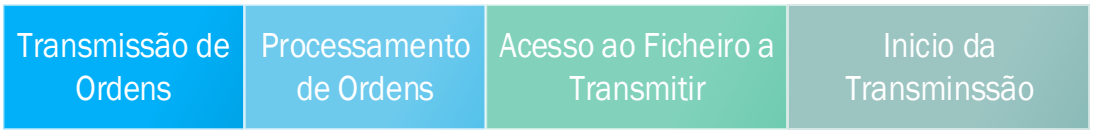

Figura 11 - Atraso ao mudar o ficheiro de som a ser reproduzido no menu

Assumindo que ambos os servidores estão na mesma rede local os únicos factores significativos de atraso são o acesso ao disco, que pode ser mitigado com o uso de armazenamento SSD, e a transcodificação que pode ser completamente evitada caso o formato em que as mensagens são armazenadas seja o meso que está a ser usado na chamada.

O acesso a gravações realizadas através da Web é feito utilizando uma base de dados onde se guarda a localização física do ficheiro da gravação em conjunto com outros dados como o utilizador a que a mensagem se destina o seu autor e a data em que foi feita. Para que isto resulte é obviamente necessário que o ficheiro seja guardado numa localização a que o servidor Kurento tenha acesso pelo que é necessário que exista algum servidor de ficheiros ou disco partilhado para que seja possível colocar o servidor Web e o servidor Kurento em máquinas diferentes (Figura 9).

4.5.3 Servidor Kurento

O servidor Kurento não requer nenhum tipo de instalação especial sendo possível instalá-lo directamente através dos repositórios da maioria das distribuições Linux. O servidor é controlado através de um WebSocket, que recebe comandos usando uma API especifica para o servidor Kurento.

Para facilitar o uso do servidor Kurento são distribuídas bibliotecas que permitem aos programadores abstraírem-se da API. Neste caso o servidor IVR foi implementado em NodeJS pelo que a biblioteca usada foi a kurento-client que é especifica de JavaScript. Esta biblioteca permite criar PlayEndpoints que reproduzem áudio e RecordingEndpoints que gravam áudio, podem ainda ser criados WebrtcEndpoints e RtpEndpoints para comunicação. Ambos os Endpoints de comunicação facilitam o envio de ofertas SDP para o servidor Kurento bem como a recepção das respostas, no caso do WebrtcEndpoint a troca de ICE candidates também é facilitada pelo Endpoint.

4.6 Implementação dos serviços

4.6.1 LogIn/Inicialização

O processo de LogIn e inicialização da aplicação Web pode ser dividido nos seguintes 11 passos presentes na Figura 12.

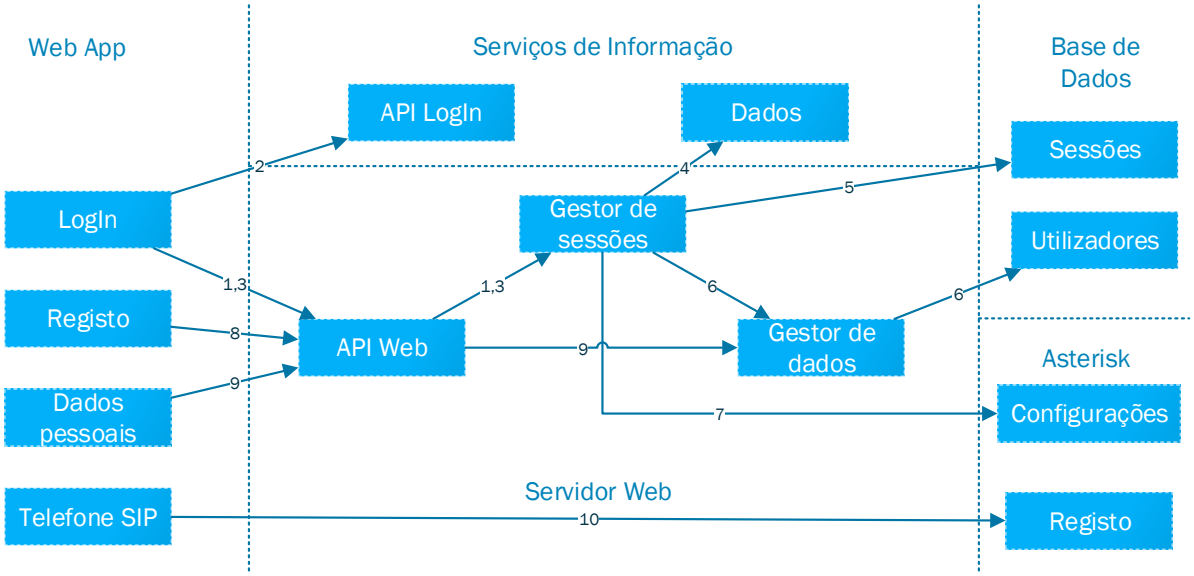

Figura 12 - Processo de autenticação/iniciação do sistema

1. A aplicação inicia o processo de Login enviando um pedido GET para a rota ***/login*** do servidor. Ao receber este pedido o servidor usa a biblioteca Oauth2 para criar um URL para o qual irá redireccionar o browser do utilizador. Este URL contém entre outras informações como o URI para o qual o cliente deve voltar depois de terminar a autenticação.
2. Redirecionado pelo servidor Web o browser do cliente acede á página de LogIn do sistema Fénix onde lhe é pedido que se autentique e no caso de ser a primeira vez é requerido que autorize ou recuse o acesso aos dados pedidos pela aplicação Web. Após a autenticação o sistema Fénix redirecciona o browser do cliente para o URI fornecido pelo servidor Web do passo 1. O sistema Fénix obriga as aplicações que se registam a fornecer este URI durante o registo, verificando mais tarde se o que tem registado é igual ao que vem nos pedidos de autenticação, caso sejam diferentes o fénix bloqueia o pedido.
3. Após estar concluído o processo de autenticação do utilizador o browser é redireccionado para a rota de autenticação do servidor. Aqui o servidor extrai o token de acesso que resultou da autenticação para preparar o passo seguinte.
4. Usando o token o servidor Web acede aos dados públicos do utilizador através da API do sistema Fénix. Após obter uma resposta o servidor verifica se contém os dados esperados, contendo os campos esperados o servidor dá utilizador como autenticado iniciando de seguida os procedimentos necessários para a fase seguinte.
5. O servidor gera um identificador aleatório (ID) para servir de chave em todos os futuros acessos, sendo por isto colocado no cookie da sessão. De seguida o servidor cria uma entrada na base de dados para a sessão contendo o ID, IstID (nome do utilizador), um espaço para o identificador de um WebSocket e a palavra chave do servidor Asterisk para este utilizador.
6. De seguida o servidor prossegue com a criação ou actualização dos dados públicos na base de dados do servidor. Se o utilizador está a usar a aplicação pela primeira vez é criada uma entrada na base dados, caso contrário a existente é actualizada com a informação mais recente.

7. Por fim o servidor verifica os ficheiros de configuração do Asterisk para criar uma extensão e utilizador caso seja necessário. Em uso normal o utilizador já se encontra registado no Asterisk sendo este passo só uma confirmação que tudo está bem, mas caso não esteja o servidor cria uma extensão para o utilizador e/ou define o seu nome de utilizador e palavra-chave, recarregando depois os módulos relevantes do Asterisk para que as alterações surtam efeito.
8. No final de uma autenticação bem-sucedida o browser é redirecionado para a aplicação Web mais especificamente para a rota ***/#!/register.*** Esta rota não tem nenhuma página Web associada, mas o seu controlador é responsável por iniciar o processo de registo da aplicação Web. Aqui a Web App envia a mensagem ***register*** para WebSocket do servidor enviando o ID no corpo da mensagem. O servidor usa o ID para aceder á sessão do utilizador e adiciona o identificador do WebSocket do cliente á informação da sessão. Após registar o WebSocket o servidor envia uma mensagem ***registred*** sem corpo para confirmar o sucesso da operação.
9. De seguida a aplicação envia um pedido GET para a rota ***/mydata*** do servidor, o servidor procura na base de dados de sessões e de dados públicos a informação do utilizador e retorna-a na resposta. Entre os dados está a palavra-chave do Asterisk que será usada para registar o telefone SIP da aplicação.
10. Por fim a aplicação usa as informações obtidas do servidor para registar o seu telefone SIP no servidor Asterisk.

    No fim deste processo a aplicação está pronta para fazer e receber chamadas através do servidor Asterisk ou directas através de instâncias desta aplicação.

4.6.2 Procura de pessoas

A procura de pessoas pode ser dividida em 3 passos como descrito na Figura 13.

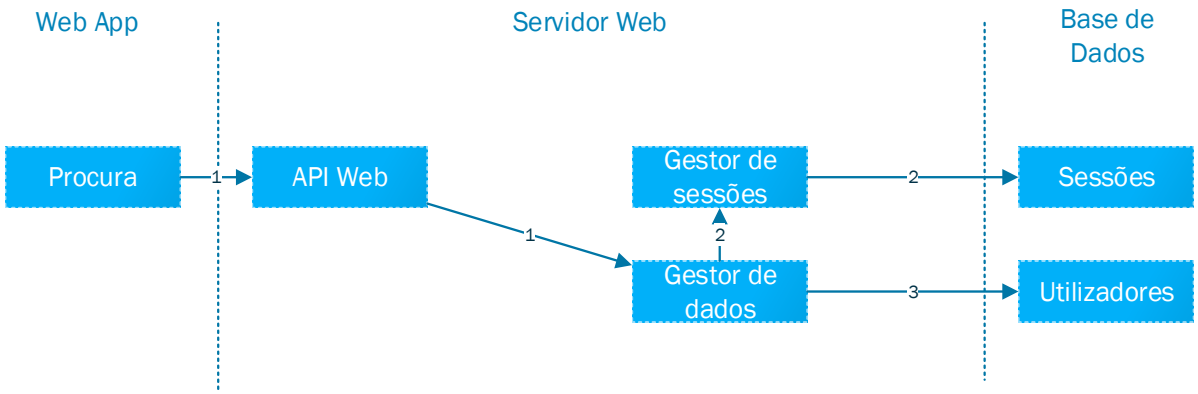

Figura 13 - Processo de procura de pessoas

O sistema permite procurar por istID ou por nome, mas uma vez que ao nível da implementação ambas as procuras são quase iguais a explicação será generalizada às duas.

1. Para iniciar uma procura a aplicação começa por enviar um pedido POST para a rota /peaple/searchby/username ou /peaple/searchby/name dependendo do tipo de procura que o utilizador pretenda fazer.

2. Quando o pedido chega ao servidor este começa por verificar se o pedido foi originado numa aplicação registada. Verificando se o identificador está na base de dados de sessões. Se estiver a operação prossegue caso contrário é abortada enviando uma resposta negativa á aplicação.
3. De seguida o servidor faz um pedido á colecção Utilizadores da base de dados usando o texto de procura fornecido pela aplicação. A procura está definida para ignorar letras maiúsculas e minúsculas enquanto procura por uma secção de texto igual ao que foi fornecido. A base de dados retorna um vector com todos os dados de todos os utilizadores que cumprem o critério da procura. O servidor retorna este vector para a aplicação terminando a procura.

4.6.3 Chamada browser para browser

Este processo de chamada pode ser dividido numa fase de preparação, que inicia um pedido de chamada e notifica o utilizador alvo, e numa outra fase de sinalização que se inicia depois da chamada ser aceite e trata de abrir os portos e negocias os codecs a usar na ligação.

A primeira fase pode ser dividida em 6 passos como descrito na Figura 14 e a segunda fase noutros 6 passos onde são trocadas mensagens no canal de sinalização também assinalado na Figura 14.

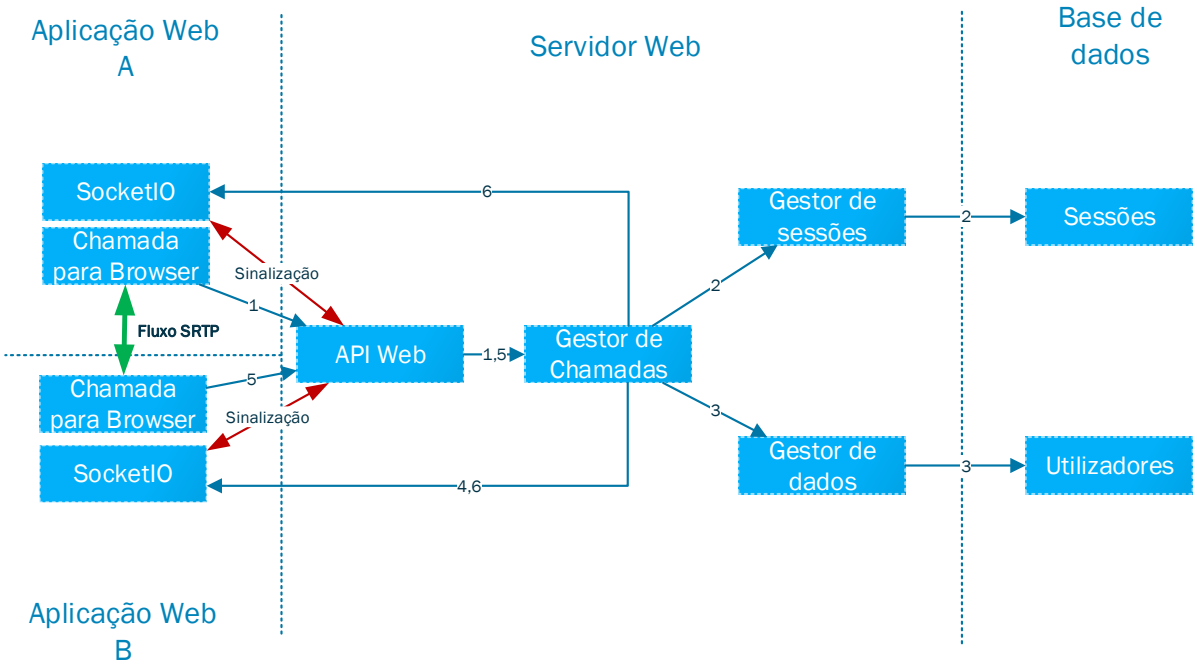

Figura 14 - Processo que leva ao estabelecimento de uma chamada peer to peer

1. O servidor dá a possibilidade de um utilizador realizar chamadas e videochamadas via WebRTC, para iniciar uma chamada basta usar a rota ***/call*** enviando o roomID associado ao cliente e o nome de utilizador da pessoa que se quer contactar.
2. Após receber o pedido o servidor verifica se utilizador está registado através do identificador presente no cookie e se estiver procura sessões activas do utilizador que se pretende contactar. Se encontrar uma ou mais sessões activas o servidor responde ao iniciador com uma mensagem de confirmação a informar que a chamada se encontra em progresso.

3. Sabendo que o utilizador se encontra registado e que o alvo da chamada tem pelo menos uma sessão activa o servidor prossegue para identificar o nome completo do utilizador que iniciou a chamada e para isto acede á colecção utilizadores da base de dados.
4. Uma vez na posse de todos os dados sobre o iniciador da chamada falta ao servidor saber se o alvo da chamada pretende ou não atender a chamada. Sendo assim o servidor envia uma mensagem através do WebSocket para todas as sessões activas do cliente. Esta mensagem contém um callRequestID, o nome completo do iniciador da chamada e o seu nome de utilizador. O callRequestID permite que ao servidor extrair da sua cache o pedido de chamada feito pelo iniciador, sendo que se este não for respondido 90 segundos após a sua criação a cache está configurada para apagar o registo e informar o iniciador que a chamada não foi atendida.
5. Se o utilizador atender ou recusar a chamada a aplicação envia um POST request para o servidor enviando de volta o objecto recebido no passo 4 adicionando apenas o parâmetro booleano ***confirm*** que será ***true*** se o utilizador aceitou a chamada e ***false*** caso contrário.
6. Ao receber a mensagem o servidor usa o callRequestID para extrair o pedido de chamada ede seguida verifica se o utilizador aceitou a chamada. Se não aceitou o servidor informa o iniciador através do WebSocket, caso tenha aceite o servidor cria uma sala usando o WebSocket SocketIO para a sinalização da chamada. Esta sala permite ao servidor redirecionar de forma rápida mensagens de uma aplicação para outra. Uma vez criada a sala as aplicações recebem o identificador da sala através do WebSocket com a mensagem ***callReady.*** A partir deste momento o processo de sinalização do ser iniciado.

O processo de sinalização está directamente relacionado com o funcionamento das API's WebRTC neste caso como foi usada a biblioteca ***adapter.js*** as interfaces usadas foram a ***RTCPeerConnection*** e a ***MediaDevices.*** Estes seguintes 6 passos ocorrem no canal de sinalização da Figura 14, assinalado com cor encarnada:

1. Quando as aplicações recebem a mensagem ***callReady*** é criado de ambos os lados da ligação o objecto PeerConnection [23] e é também usada a função ***getUserMedia*** disponibilizada pela interface ***MediaDevices*** [24] para obter o som e se requisitado a imagem da câmara (stream). Assim que a stream está pronta esta é adicionada ao objecto PeerConnection.
2. No lado do iniciador da chamada quando a stream é adicionado é o objecto PeerConnection é usado para gerar uma oferta, esta oferta é escrita seguindo o protocolo SDP e descreve a stream a ser transmitida na sessão que irá ser estabelecida bem como vários parâmetros da sessão. Esta oferta é enviada através do SocketIO usando o canal de sinalização que foi estabelecido. Após gerar esta oferta a PeerConnection começa automaticamente a gerar candidatos ICE.
3. Ao receber a oferta vinda do iniciador da chamada a aplicação adiciona-a ao objecto PeerConnection. Nesta fase ou a aplicação cria uma resposta caso já tenha adicionado a sua stream local ou caso não tenha espera que a sua stream seja adicionada. Uma vez que é necessária a expressa autorização do utilizador para que uma aplicação Web tenha acesso á câmara é muito comum o processo de iniciar a câmara levar largos segundos. Podendo a oferta do par chegar muito antes da stream local estar pronta.

4. Tendo gerado a resposta a aplicação alvo da chamada usa o canal de comunicação para a enviar para o iniciador. A partir deste momento esta aplicação também inicia a recolha dos candidatos ICE.
5. Ambas as aplicações enviam através do canal de comunicação os candidatos ICE á medida que os vão gerando implementando, portanto ***Trickle ICE***. Os candidatos são também adicionados aos objectos PeerConnection á medida que são recebidos.
6. Quando ambas as aplicações encontram candidatos ICE que são válidos iniciam-se os fluxos SRTP estre os dispositivos/aplicações e podemos dizer que a chamada está estabelecida, podendo ainda assim ser transferida para outro endereço de transporte caso este se venha a revelar vantajoso para a ligação.

4.6.4 SoftPhone para Browser

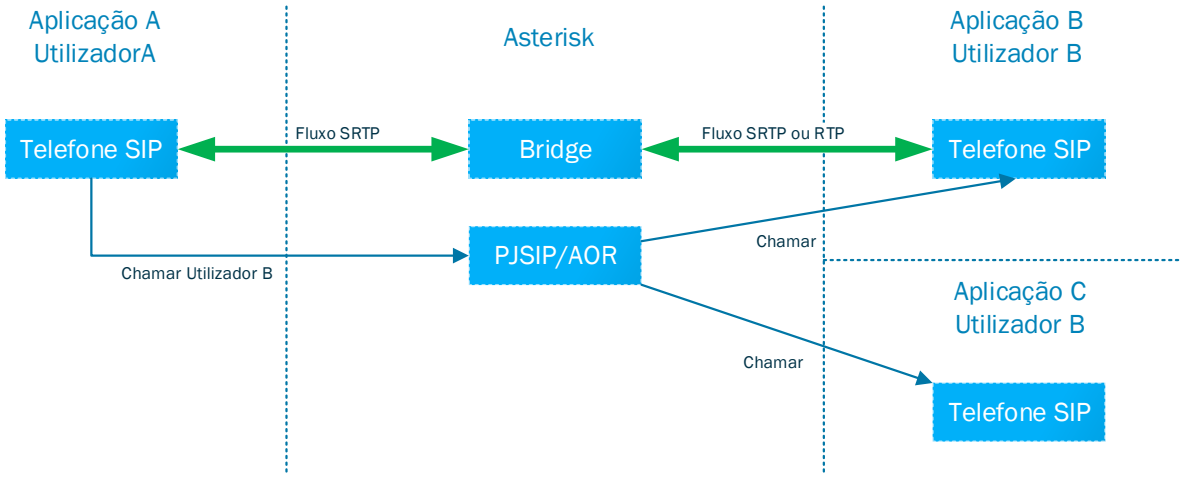

Figura 15 - Processo para o sistema estabelecer uma chamada SIP

Neste sistema as chamadas SIP são feitas sem nenhuma intervenção do servidor Web, como se pode ver na Figura 15, sendo que a comunicação é feita directamente entre a aplicação Web e o servidor Asterisk. Para fazer uma chamada a aplicação apenas precisa de saber o número SIP que se quer contactar. A biblioteca JSSIP gere todo o processo de ligação sendo apenas necessário fornecer os elementos HTML onde se quer reproduzir as streams locais e remotas.

O servidor Asterisk está configurado para contactar todos os dispositivos associados com o utilizador podendo qualquer um responder á chamada.

As chamadas via Asterisk usam o protocolo SIP para o registo de dispositivos e para o estabelecimento e controlo de sessões. O protocolo SDP é usado para negociar os codecs e outras características da sessão como os ritmos de transmissão.

Se for necessário realizar transcodificação por dois terminais não suportarem nenhum codec comum o Asterisk encarrega-se automaticamente da tarefa.

Os fluxos de dados entre o browser e o servidor Asterisk são sempre encriptados com SRTP, mas os fluxos entre o Asterisk e outro telefone SIP qualquer podem não ser. Para garantir a confidencialidade

das comunicações é aconselhável que todas as secções da chamada sejam encriptadas, mas se o telefone SIP não suportar SRTP e a rede entre o servidor e o telefone for segura esta configuração é uma opção viável.

4.6.5 Acesso ao gravador no browser

O servidor Web permite a entrega de mensagens de áudio ou vídeo, através de upload de ficheiros. Para deixar uma mensagem basta usar a rota ***/record*** um POST com o utilizador destino e um blob contendo os dados do ficheiro, estes dados devem estar dentro de um objecto FormData.

O servidor Web usa o pacote Multer para processar o FormData recebido e guarda o ficheiro em disco para acesso futuro. É criada uma entrada na base de dados com o nome do utilizador que gravou a mensagem, o utilizador a que é destinada, caminho absoluto onde está o ficheiro guardado e o identificador da mensagem.

O servidor permite o acesso aos ficheiros guardados aos utilizadores registados através da rota ***/record/:filename*** utilizando o método GET.

A gravação pode ser apagada usando a rota ***/record/messageID*** com o método DELETE, bastando ser apenas o utilizador que criou a mensagem o destinatário da mensagem.

4.6.6 Acesso ao IVR do gravador (POTS e SoftPhone)

O sistema IVR dá a todos os utilizadores que usem um telefone SIP ou POTS a possibilidade de deixarem mensagens de voz a outros utilizadores usando para isto o sistema IVR. Na realidade neste sistema o gravador e o sistema IVR acabam por ser a mesma coisa isto porque o gravador é um exemplo de um menu IVR criado usando o sistema IVR.

Assim sempre que um utilizador não atende uma chamada os utilizadores são redireccionados para o sistema IVR que podem utilizar para gravar uma mensagem.O redireccionamento é feito pelo Asterisk sempre que uma chamada não é atendida porque o tempo expirou ou é rejeitada. Ao receber o redireccionamento da chamada o servidor IVR inicia uma chamada entre o servidor Kurento e o Endpoint do Asterisk que realizou o redireccionamento, comandado de seguida o servidor Kurento a reproduzir a mensagem de voz que explica ao utilizador o que deve fazer se quiser deixar uma mensagem.

Se o utilizador estiver interessado em deixar uma mensagem o servidor IVR comanda o Kurento de modo a que este comece a gravar a stream de áudio que está a receber para um ficheiro. Concluída a gravação, que é confirmada quando o servidor IVR recebe um sinal DTMF específico originado pelo cliente, o servidor Kurento é comandado para parar a gravação. De seguida o servidor IVR adiciona os meta-dados da nova gravação á base de dados para que seja possível que no futuro o servidor IVR e o servidor Web possam encontrar esta gravação e entregá-las aos clientes. Na base de dados é gravado o caminho absoluto para o ficheiro de áudio, o nome do autor da mensagem a data e hora a que foi deixada e o utilizador a que se destina.

Para que o sistema IVR consiga saber a quem se destina uma dada mensagem é necessário que o servidor Asterisk envie explicitamente o identificador do utilizador destino. Isto é um problema visto que no redireccionamento, os cabeçalhos SIP que o servidor IVR recebe não contêm informação nenhuma sobre o original destinatário da mensagem. De modo a resolver este foi criada uma macro no Asterisk que é passada na função ***Dial*** do dialplan, a macro é simples e faz apenas com que o servidor Asterisk envie através de DTMF's o identificador do utilizador a que a chamada originalmente se destinava. Assim o menu de gravação deve estar pronto para receber 6 números (que correspondem á extensão do utilizador) como DTMF's assim que a chamada tem início. Só depois destes 6 dígitos é que o utilizador tem a hipótese de interagir com o sistema.

Para um utilizador ouvir as mensagens que lhe foram enviadas o utilizador apenas tem de ligar para uma extensão especifica do servidor IVR á qual tenha sido atribuído um menu que realize essa tarefa. Para um menu recuperar todas as mensagens que tenham sido enviadas para um utilizador apenas é necessário fazer um simples ***find*** na base de dados mongoDB usando o nome de utilizador que está a contactar o sistema IVR (disponível no identificador SIP) como critério de busca. Se existirem gravações na base de dados o Sistema IVR reproduz uma mensagem (usando o Kurento) de áudio com instruções sobre que teclas premir para poder ouvir todas as mensagens.

É de notar que este sistema é capaz de reproduzir as mensagens de áudio e vídeo que foram deixadas usando o servidor Web embora no caso dos vídeos apenas o áudio seja transmitido. O mesmo é válido para o servidor Web sendo capaz de entregar as mensagens que foram gravadas com recurso a sistema IVR da mesma forma como entrega as mensagens de áudio que foram gravadas usando a Web.

4.6.7 Edição de directório

O servidor permite o armazenamento de números de telefone SIP como alternativas de contacto. Para gerir esta funcionalidade o servidor tem três rotas definidas. Para criar um número o utilizador deve usar um POST na rota ***/number*** enviando o número na mensagem, isto adiciona um novo número na base de dados associado ao utilizador. Para obter todos os números de um utilizador basta apenas usar a rota ***/number/:usename*** com o método GET, o servidor retorna todos os números associados com o utilizador. Por último para remover um número o servidor define a rota ***/number/:number*** que usando o método DELETE permite a um utilizador apagar os números que definiu.

5 Resultados / Validação

5.1 Possibilidades de comunicação

Tabela 4 - Comunicações possíveis entre os diferentes sistemas

Iniciador da chamada	Destino da chamada				
	POTS	**Softphone**	**WebRTC**	**IVR**	**Gravador**
POTS	Sim	Sim (1)	Sim (1)	Sim (1)	Sim (1)
Softphone	Sim	Sim	Sim (2)	Sim (1)	Sim (1)
WebRTC	Sim (1)	Sim (2)	Sim (2)	Sim (1)	Sim (3)

(1) – Sim através da redirecção do servidor Asterisk

(2) – Sim e de forma directa

(3) – É feito através do upload de um ficheiro

O sistema montado permite realizar chamadas bidireccionais entre todos os sistemas presentes como se pode ver na tabela 4. As comunicações entre clientes WebRTC são feitas de forma directa, bem como as comunicações entre softphones e clientes WebRTC desde que estes usem configurações compatíveis o Asterisk tenta sempre que os clientes comuniquem directamente. Infelizmente com as redes POTS, IVR e o gravador precisam sempre de ser redirecionadas pelo Asterisk devido a incompatibilidades de configuração, excepção feita á relação entre WebRTC e o gravador por não utilizar o IVR de gravação, mas sim um serviço Web específico para deixar mensagens.

5.2 Funcionamento da aplicação Web

Esta secção destina-se a explicar e demonstrar com imagens o funcionamento de todas as funcionalidades que a aplicação Web oferece indicando para cada página os endpoints da API web que são necessários invocar para compor a página ou para realizar uma qualquer operação.

Antes de começar a utilizar a aplicação é necessário visitar estes dois links e autorizar os seus certificados.

- https://146.193.41.162:8443/ - Servidor Web
- https://146.193.41.162:8089/httpstatus - Servidor Asterisk

O primeiro link corresponde ao servidor Web que disponibilizada a Web App, enquanto que o segundo é uma página web servida pelo servidor Asterisk usada para diagnosticar o estado do servidor Web do Asterisk, estapágina pode ser vista na Figura 16. Esta segunda autorização é necessária para que a aplicação web possa aceder ao WebSocket através de WSS no mesmo IP e porto. Estes passos só são necessários pelo facto de os certificados não serem assinados por uma autoridade certificadora. Depois destes dois passos a aplicação está pronta a ser usada.

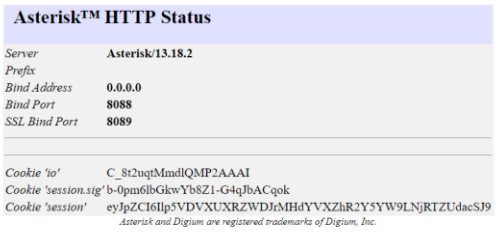

Figura 16 - Página do servidor Asterisk

5.2.1 LogIn

Quando o utilizador abre a aplicação sem estar autenticado o que aparece é uma página como a que se vê na Figura 17.

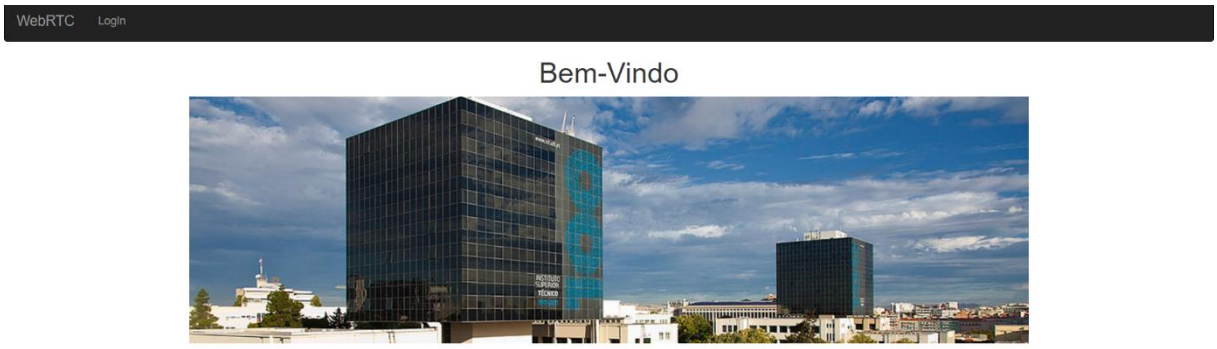

Figura 17 - Página inicial da aplicação Web

O processo de login é simples para o utilizador assim que o utilizador carrega no botão de login é redirecionado para a página de autenticação do sistema fénix. No primeiro login o utilizador é questionado pelo sistema fénix se deve dar autorização á aplicação para aceder aos seus dados públicos. Para que o sistema funcione é necessário que o utilizador autorize este acesso.

Figura 18 - Página inicial após LogIn

Quando o LogIn no sistema Fénix termina o browser é redireccionado para a página principal que agora que o utilizador está autenticado tem o aspecto mostrado na Figura 18, a partir deste momento a aplicação está pronta a receber chamadas do servidor Asterisk ou de outro browser.

5.2.2 Página pessoal

Cada utilizador tem uma página pessoal onde pode ver os seus dados os números que tem associados e as mensagens que recebeu de outros utilizadores, o aspecto desta página pode ser visto na Figura 19.

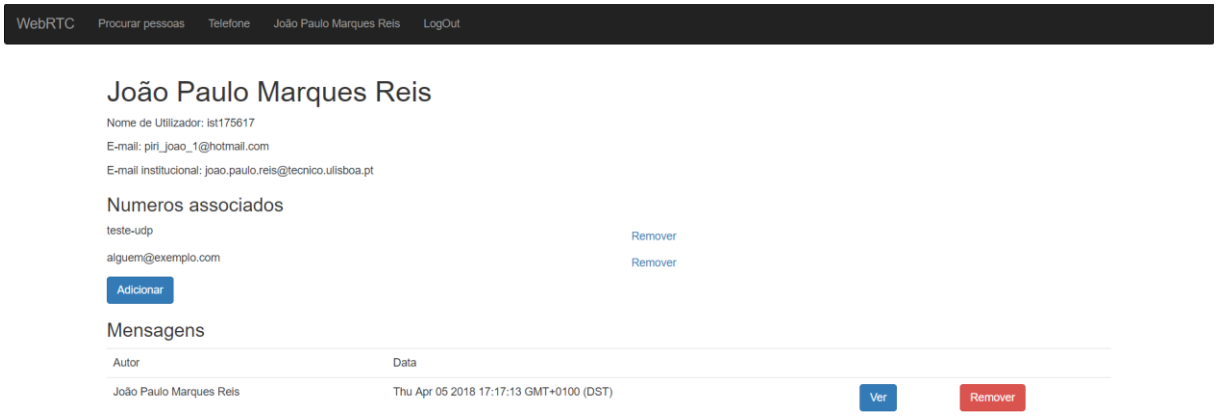

Figura 19 - Página pessoal

Os dados que se podem ver são o nome completo o nome de utilizador e os e-mails.

O botão **"Adicionar"** permite associar números SIP ao utilizador para que os outros utilizadores o possam contactar através desses números. A página que contém várias directivas Angular para gerar código HTML dinamicamente, gera para cada número SIP associado um botão que o permite remover.

A lista de mensagens tal como os números SIP é gerada dinamicamente com e tem um botão para apagar a mensagem e outro para a ouvir/ver.

Quando o utilizador escolhe ver uma mensagem a aplicação redireciona o utilizador para outra página onde o elemento HTML5 vídeo usa a rota do servidor ***/record/:filename*** para adquirir a mensagem.

5.2.3 Procura de utilizadores

Procurar utilizadores é feito através da opção "Procurar pessoas" disponível na rota ***/search***, aí é disponibilizada uma página com uma form HTML onde o utilizador pode escolher se pretende procurar por nome de utilizador ou pelo nome. A form tem também uma caixa de texto para escrever a String a procurar como se pode ver na Figura 20.

Figura 20 - Página para procurar outros utilizadores

Depois de clicar no botão procurar a aplicação envia um pedido para servidor para a rota ***/searchby/username*** ou para a rota ***/searchby/name*** dependendo das opções escolhidas. O pedido é feito utilizando o módulo **$http** do AngularJS. O resultado do pedido deverá ser um vector contendo os dados dos utilizadores que cumprem os critérios da procura. Após receber o resultado do servidor a aplicação troca para a rota ***/search/result*** carregando novo código HTML, passando a página a ter o aspecto mostrado na Figura 21.

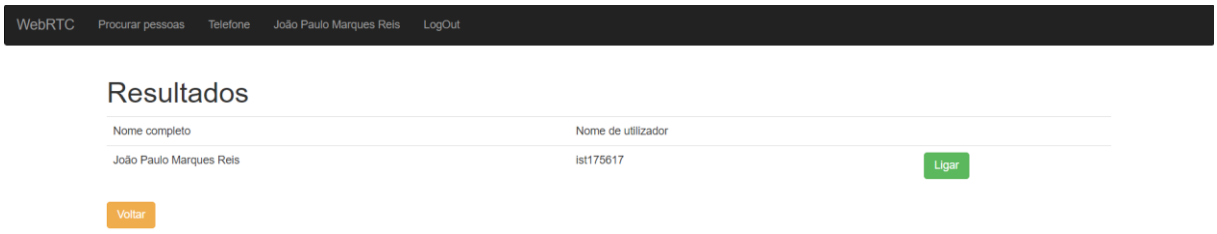

Figura 21 - Lista com resultados de uma procura

Para além de novo código HTML uma nova rota traz também um novo controlador com variáveis novas, assim para passar o resultado da procura do controlador antigo para o novo foi necessário usar um serviço.

Este serviço tem apenas um atributo e dois métodos sendo estes o “get” e o “set”, este serviço guarda no atributo o que quer que tenha sido passado como argumento para o “set” pela última vez enquanto que o “get” apenas retorna o valor do atributo. Uma vez que serviços Angular criados através de fábricas são persistentes não dependem dos controladores os valores que lá são colocados podem ser acedidos por qualquer controlador.

O novo código HTML contém directivas do Angular para poder gerar dinamicamente o código necessário para mostrar todos os resultados que o servidor retornou. Para além disso também contem

dois botões que um para voltar á página anterior e fazer outra procura e outro para fazer uma chamada para o utilizador.

5.2.4 Chamadas

A aplicação oferece duas formas de fazer chamadas, através do resultado da procura de pessoas onde se pode usar o botão **"Ligar"** escolher o número do utilizador para o qual se quer ligar, como se pode ver na Figura 22.

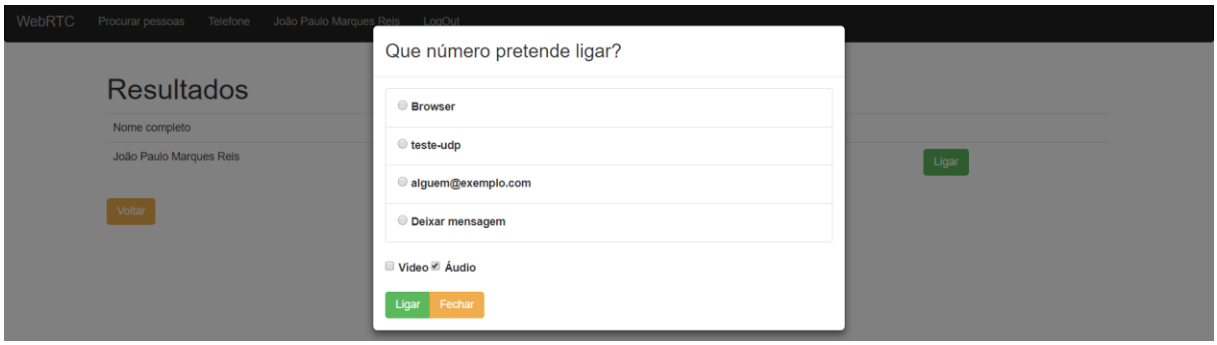

Figura 22 - Lista com as opções para contactar o utilizador

Outra opção é usar a opção **"Telefone"** na barra de navegação e introduzir manualmente o número SIP que pretende contactar, usando a interface de utilizador da Figura 23.

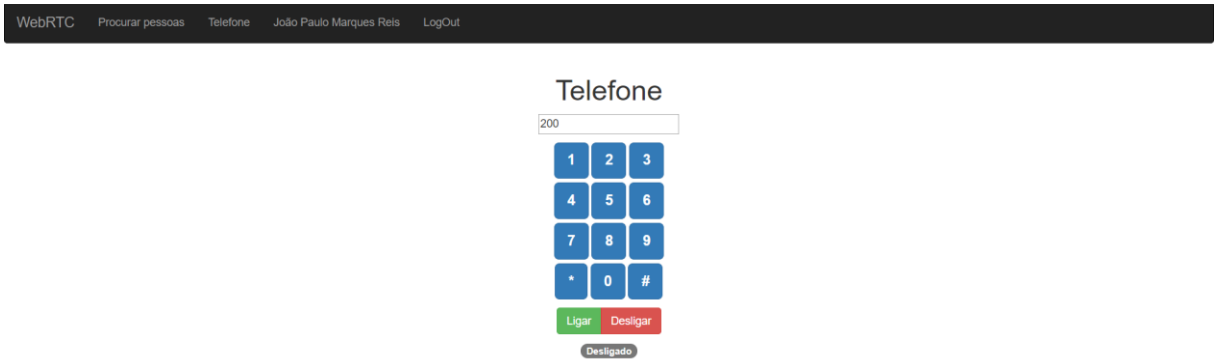

Figura 23 - Página do telefone SIP

Usando o sistema de procura é apresentado ao utilizador uma lista de números e formas de contacto. A lista mostra uma opção para ligar directamente para o browser do outro utilizador, caso este esteja online, mostra também a lista de números SIP que o utilizador associou á sua conta e por fim mostra uma opção para deixar uma mensagem de voz ou vídeo.

Para criar esta lista de opções a aplicação usa duas rotas que estão definidas no servidor fazendo dois pedidos GET independentes. Um para verificar se o utilizador está online e outro para obter a lista dos números que estão associados ao utilizador. Uma vez que são independentes as operações são realizadas em paralelo.

Caso o utilizador escolha contactar um browser a aplicação usa a rota do servidor ***/call*** para contactar o outro utilizador. Quando a chamada é atendida a aplicação é muda automaticamente para uma página

com dois elementos de vídeo HTML5, uma para ver a imagem da própria câmara e outro para ver a imagem remota, como se vê na Figura 24.

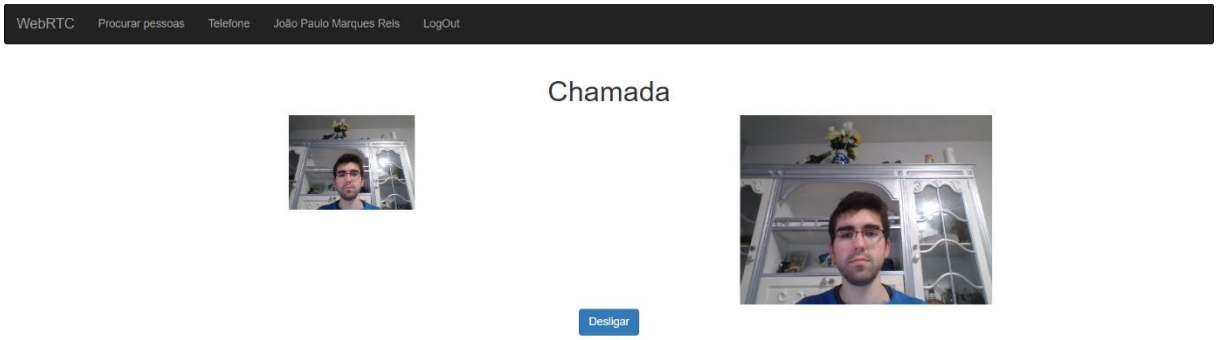

Figura 24 – Chamada entre dois browsers com vídeo

Caso o utilizador escolha um número SIP a aplicação passará imediatamente para a página do **"Telefone"** e iniciará de imediato o contacto, como se pode ver na Figura 25 o softphone notifica de imediato o utilizador da chamada que está a receber e a aplicação mostra o estado da chamada como estando em progresso.

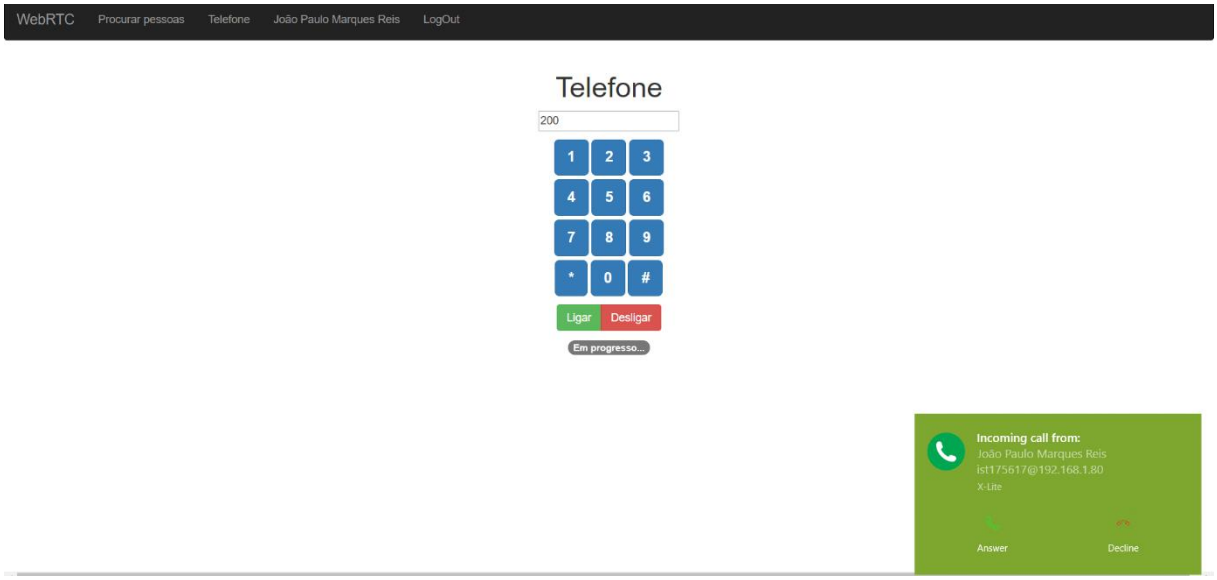

Figura 25 - Telefone SIP a contactar um número de teste

As chamadas usando SIP e o servidor Asterisk e as chamadas de browser para browser foram implementadas na aplicação em serviços distintos do Angular sendo a sua expansão e manutenção independente.

A página da aplicação para chamadas SIP (Figura 23) tem os botões que é normal encontrar em qualquer telefone e podem ser usados para enviar DTMF's para um IVR, ou outro qualquer sistema do género. Usando o botão de ligar a chamada é feita para o número que aparece na caixa de texto acima dos números.

A chamada browser para browser usa a API fornecida pela biblioteca Adapter.js para criar um módulo compatível com vários browsers. A aplicação foi testada nos nas versões mais recentes dos browsers Firefox e Chrome, tendo completa compatibilidade com estes, em ambas as versões Windows e Android. O Microsoft Edge apresenta vários problemas ao nível da implementação do projecto WebRTC apresentando ainda uma grande instabilidade, por este motivo a aplicação não suporta o browser Edge.

O módulo que gere as comunicações com o servidor Asterisk, também recorre ao Adapter.js para gerar as Streams de áudio e vídeo locais. Todo o resto é implementado em JavaScript, pelo que também para estas funcionalidades os browsers Firefox e Chrome são compatíveis. O Edge apresenta os mesmos problemas.

5.2.5 Deixar mensagem a outro utilizador

A aplicação permite que o utilizador grave uma mensagem e a envie para outro. Esta funcionalidade recorre a WebRTC para obter os dados da câmara e microfone guardando depois os dados que são obtidos num vector.

Quando o utilizador escolhe ligar para outro utilizador a aplicação dá-lhe a opção de deixar uma mensagem, aí o utilizador pode escolher se quer deixar uma mensagem de voz ou de vídeo. Quando o utilizador conforma as escolhas a aplicação muda para a página de gravação.

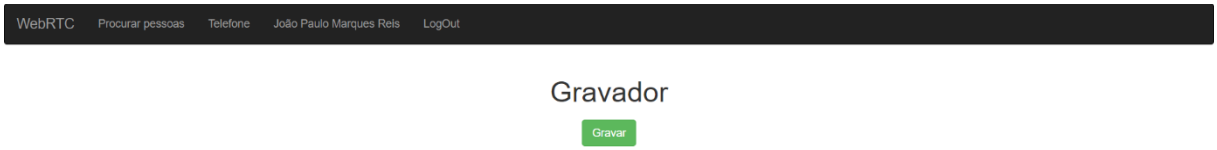

Figura 26 - Aspecto inicial da página do gravador

Inicialmente é apresentada ao utilizador uma página com apenas um botão para iniciar a gravação da mensagem como mostra a Figura 26.

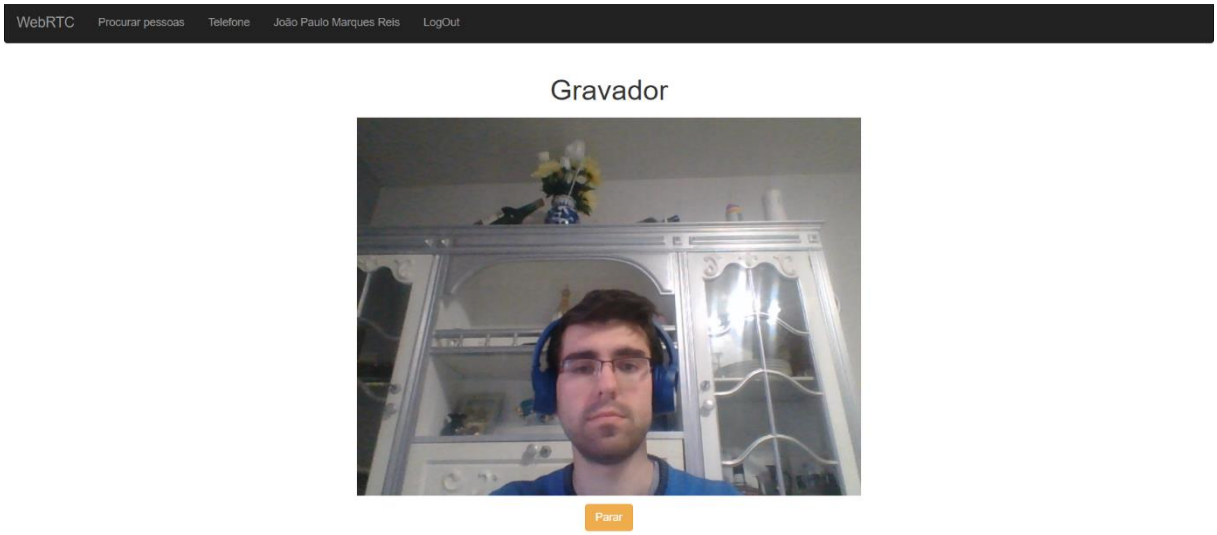

Figura 27 - Gravador durante a gravação

Após carregar no botão para gravar a gravação da mensagem inicia-se o botão **"Gravar"** desaparece e dá lugar a um botão para parar a gravação. Quando se está a captar vídeo a aplicação mostra o que está a ser gravado no ecrã através de elemento de vídeo HTML5, como se vê na Figura 27.

Ao iniciar a gravação a aplicação usa a API do projecto WebRTC para iniciar os dispositivos de gravação e criar uma stream de áudio/vídeo. Assim que a stream está criada a aplicação usa a interface MediaRecorder [25] para gravar os dados que são colocados na stream pelos dispositivos. Esta interface só está disponível nos browsers Firefox Chrome pelo que esta funcionalidade está restringida a estes browsers. O MediaRecorder activa um evento sempre que existem novos dados na stream, passando esses mesmos dados para o handler do evento, a aplicação limita-se a colocar os dados num vector, chamado **"recordedChunks",** para uso futuro.

Figura 28 - Gravador após a paragem da gravação

Quando o utilizador decide interromper a gravação a aplicação mostra três botões com opções, como mostra a Figura 28. Usando a interface o utilizador pode:

- Escolher gravar outra mensagem descartando a primeira;
- Submeter a gravação que fez enviando-a para o servidor;
- Visualizar a gravação que fez para decidir se envia ou não.

Quando o utilizador decide parar a gravação a aplicação pára a gravação através da interface MediaRecorder. Esta acção faz com que o evento de **"onstop"** seja disparado executando o handler que cria um Blob a partir do vector recordedChunks. Depois usando o Blob cria um ObjectURL para permitir visualizar a gravação caso o utilizador o deseje.

Se um utilizador escolher visualizar a gravação que acabou de fazer a aplicação usa o objectURL criado como fonte num elemento HTML de vídeo, o resultado é o mostrado na Figura 29.

Figura 29 - Previsão da gravação feita

Após a visualização da imagem ainda estão disponíveis as mesmas opções sendo que se o utilizador clicar outra vez no botão **"Previsão"** o vídeo começa de novo.

Quando o utilizador decide enviar a mensagem a aplicação cria uma FormData na qual coloca o Blob e o nome de utilizador do destino. Esta FormData é depois colocada no corpo de um POST request que é enviado para a rota **"/record"**. Quando o servidor responde o utilizador é informado que a mensagem foi enviada com sucesso como se mostra na Figura 30.

Figura 30 - Gravador após enviar a gravação para o servidor

5.3 Menus IVR disponíveis

Para melhor demonstrar o funcionamento do sistema IVR foram criados dois menus IVR, um para gravar mensagens de voz e deixá-las a um utilizador e outro para ouvir as mensagens que foram deixadas. Estes menus procuram exemplificar e demonstrar o potencial de um sistema que permite desenvolver menus IVR usando uma linguagem como JavaScript em NodeJS.

5.3.1 Gravador

A única forma de aceder ao gravador IVR é através de uma chamada para outro utilizador, se este não atender a chamada é então redireccionada para o gravador. O gravador atende de imediato a chamada e recebe um número que representa a extensão para a qual o Asterisk tentou ligar sem sucesso, de seguida o gravador pede ao utilizador que pressione o número 1 para iniciar a gravação, e que pressione 1 novamente para a terminar. Depois de gravada a mensagem pode ainda ser consultada

por quem a gravou ao premir o botão 2. Este menu não permite aos utilizadores apagar a mensagem mesmo que não estejam satisfeitos como resultado, mas tal funcionalidade é possível de implementar sem grandes alterações no menu actual. Depois de gravada a mensagem pode ser imediatamente consultada pelo destinatário através do browser e do serviço de voice-mail. Na Figura 31 pode ver-se o fluxograma do menu do gravador.

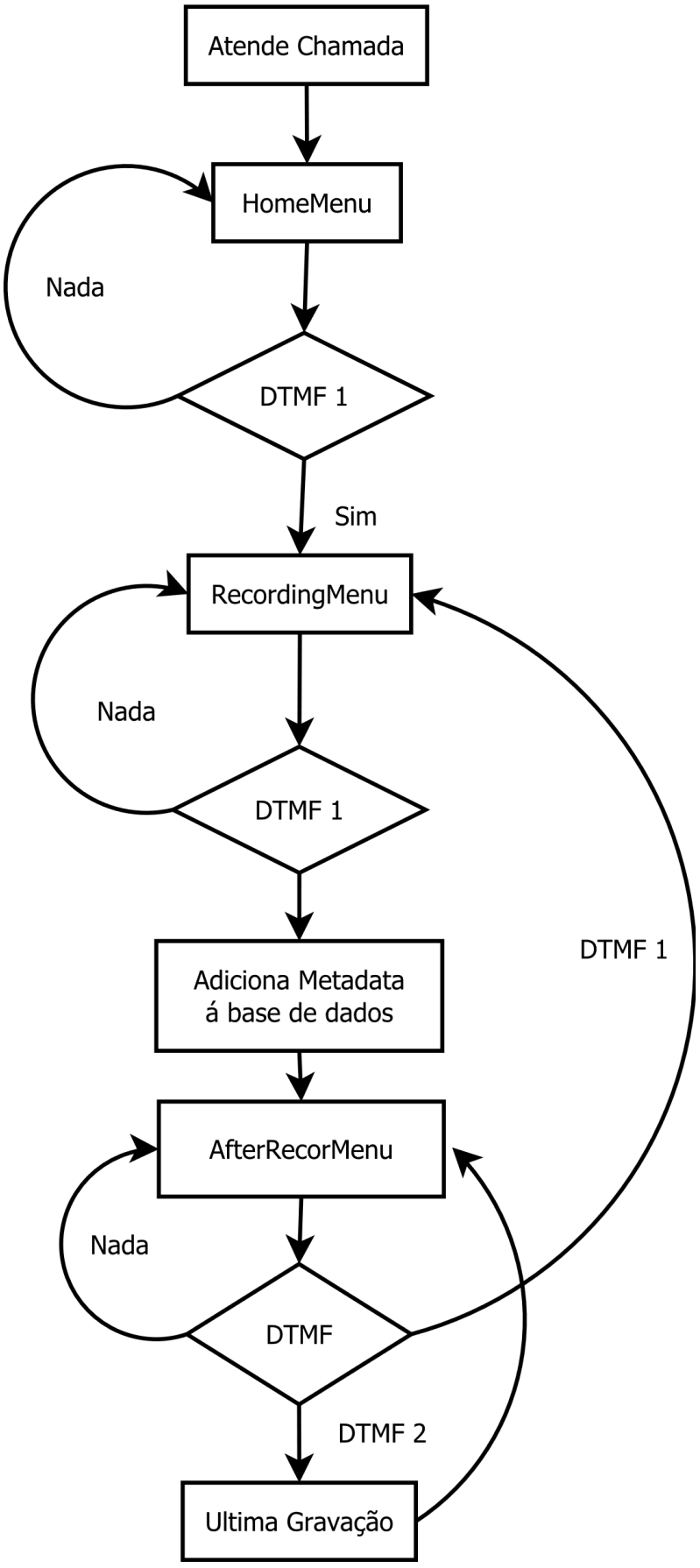

Figura 31 - Fluxograma do menu IVR do gravador

Abaixo encontra-se o código JavaScript do menu gravador

```
exports.init = function (session) {
  // Recepção caller id
  session.data.user = remoteIdentity;
  // Carrega o handler que lida com os DTMF
  session.on("dtmf", function (request, dtmf) {
    let tone = menuFunctions.parseDTMF(request);
    session.data.menu.activeMenu(session, tone);
  });
  session.accept();
  // Estado inicial
  session.data.menu.changeMenu(recordMenu);
  homeSounds(session);
};
const homeSounds = (session) => {
  const menu = session.data.menu;
  // Repete mensagem enquanto nao houver dtmf
  menuFunctions.createPlayerEndpoint(session, "Sounds/Bem-vindo-record.webm",
homeSounds);
};
const afterRecordSounds = (session) => {
  menuFunctions.createPlayerEndpoint(session, "Sounds/After-Record.webm",
afterRecordSounds);
}
const recordMenu = function (session, tone) {
  if (tone == 1) {
    menuFunctions.createRecordEndpoint(session);
    session.data.menu.activeMenu = recordingMenu;
  } else if (tone == 2) {
    menuFunctions.createPlayerEndpoint(session, session.data.recordPath,
(session) => {
      session.data.menu.changeMenu(recordMenu);
      afterRecordSounds(session);
    });
  }
};
const recordingMenu = async function (session, tone) {
  if (tone == 1) {
    session.data.recordEndpoint.stop() //Para Gravação
    await menuFunctions.createRecord(session); // Coloca gravação na Base de
dados
    session.data.menu.changeMenu(recordMenu); //Volta para o Record Menu
    afterRecordSounds(session); // Reproduz o som que é suposto após a criação
da primeira gravação
  }
}
```

Usando o sistema e as bibliotecas desenvolvidas neste trabalho é possível criar um menu de voicemail simples que adiciona as gravações a uma base de dados MongoDB usando apenas uma página de código JavaScript.

5.3.2 VoiceMail

O voice-mail está disponível na extensão 1000 e pode ser consultado por qualquer utilizador. Assim que o utilizador entra no voice-mail é indicado que este pressione um para começar a ouvir as mensagens. Após pressionar 1 o utilizador ouve a primeira mensagem, assim que esta termina são apresentadas 4 opções:

1. Ouvir outra vez
2. Ouvir a próxima
3. Ouvir a anterior
4. Apagar a actual

Quando o utilizador chega ao fim das mensagens é lhe dado conta disso mesmo através de uma instrução de voz e a chamada termina. Á semelhança do gravador as acções praticadas no voice-mail também têm reflexo na aplicação Web, ou seja, uma mensagem apagada no voice-mail também é apagada da aplicação (a base de dados é a mesma para os dois como mostrava a Figura 9).

5.4 Validação

Para validar algumas das escolhas feitas neste trabalho o mesmo foi apresentado aos responsáveis pelos serviços de modo a procurar verificar a sua utilidade.

As funcionalidades analisadas foram a configuração automática que o servidor Web realiza sobre o Asterisk de forma a adicionar utilizadores, e o sistema IVR criado de forma a permitir o desenvolvimento de menus IVR usando NodeJS.

A configuração dos utilizadores foi bem recebida uma vez que suprime uma necessidade do Asterisk permitindo criar utilizadores através de uma API REST do servidor Web.

A ideia do servidor IVR também foi bem recebida uma vez que abre a possibilidade de criar menus usando uma tecnologia familiar a um grande número de trabalhadores dos serviços informáticos o que garante que a manutenção e expansão do código pode ser feita sem depender apenas de único especialista que aprenda a funcionar com o Asterisk. Além da conveniência da manutenção o menu IVR em NodeJS permite ainda integrar os menus IVR com serviços Web abrindo todo um novo leque de novos serviços que é possível oferecer aos utilizadores.

6 Discussão

6.1 Segurança

6.1.1 Comunicações com o servidor Web

As comunicações entre os browsers e o servidor Web são todas efectuadas através dos protocolos HTTPS e WSS, sendo que estes protocolos partilham o mesmo socket e por isso o mesmo nível de encriptação SSL (Secure Socket Layer [26]). É necessário fornecer ao servidor Web a chave privada e o certificado devendo este usar uma hash de assinatura que seja considerada segura pelos browsers como o SHA256. O uso de tecnologia HTTPS em todas as mensagens trocadas garante a integridade e a confidencialidade das comunicações entre o servidor Web e as aplicações dos utilizadores.

A integridade das comunicações entre os clientes é garantida pela implementação WebRTC feita pelo browser e por uma encriptação das mensagens de sinalização através do uso do protocolo WSS. Infelizmente a implementação do protocolo SRTP, usado pelo WebRTC, não permite uma confidencialidade completa pelo que um hacker que esteja a escutar uma chamada entre dois utilizadores pode claramente identificar os intervenientes na chamada. Isto deve-se ao facto dos cabeçalhos dos pacotes RTP estarem expostos quando se usa o protocolo SRTP. Este cabeçalho contém os níveis de áudio o que na prática revela se um utilizador está a falar [27]. Ainda assim o protocolo SRTP torna impossível a um atacante perceber o conteúdo da conversa garantindo um nível razoável de confidencialidade. Na Figura 32 é possível ver a pilha de protocolos usada numa aplicação WebRTC bem como uma representação visual da exposição dos cabeçalhos dos pacotes SRTP.

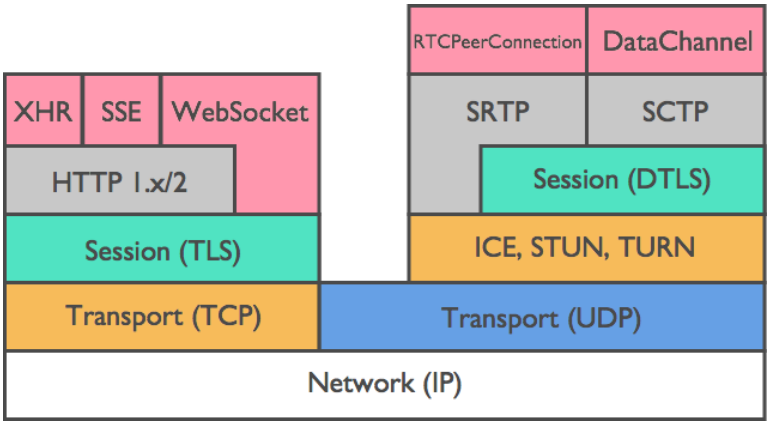

Figura 32 – Pilha de protocolos de uma aplicação WebRTC [27]

6.1.2 Autenticação do utilizador no servidor Web

A autenticação no servidor Web é feita usando o protocolo OAuth 2.0 [21], este protocolo permite confirmar a identidade de um utilizador e ao mesmo tempo obter de uma fonte de confiança os necessários ao funcionamento da aplicação.

Para facilitar o uso do protocolo foi usado o pacote Simple-Oauth2 [28] disponível para NodeJS. Este pacote permite criar de forma muito simples o URI usado para reencaminhar o utilizador para o servidor

de autorizações. O pacote usa o identificador e o segredo fornecidos pelo servidor de autorizações durante o registo da aplicação para criar o URI de redireccionamento no formato certo. O URI de redireccionamento também contém um URI a identificar o ponto da aplicação para onde o utilizador deve ser redirecionado após o processo de autenticação.

Quando o utilizador termina a autenticação e acede á aplicação novamente usando o URI do género *https://exemplo.pt/auth*, o servidor recebe também um código de autorização que foi gerado pelo servidor de autorizações. Este código é então usado pelo pacote Simple-Oauth2 para contactar o servidor de autorizações pedindo um *access token* e um *refresh token*. O *access token* é usado para pedir dados pertencentes ao utilizador usando a API do servidor detentor dos recursos enquanto que o *refresh token* serve para pedir um novo *access token* quando o mesmo expira.

Uma vez que este sistema não possui meios para autenticar um utilizador o *refresh token* é descartado. Se o sistema possuísse o seu próprio sistema de autenticação seria aconselhável usar o *refresh token* para que o utilizador não passasse pelo incómodo de se autenticar duas vezes em dois serviços distintos.

6.1.3 Comunicações com o Asterisk

Á semelhança do servidor Web o Asterisk também possui um WebSocket, o que lhe confere a possibilidade de usar o protocolo WSS para comunicar. O uso do protocolo WSS permite ao servidor Asterisk receber de forma segura mensagens SIP que estariam de outra forma desprotegidas e susceptíveis de serem lidas ou até mesmo alteradas por um atacante.

Para encriptar os fluxos de dados da chamada o Asterisk usa o protocolo DTLS (Datagram Transport Layer Security [29]) tal como na comunicação directa entre browsers o protocolo SRTP é usado sobre a camada DTLS, infelizmente os problemas identificados na secção 6.2.1 também ocorrem nesta ligação.

O Asterisk também define um método de transporte TLS para receber sinalização de outros dispositivos. As sessões criadas usando este método devem também encriptar as suas sessões para garantir o mesmo nível de segurança das comunicações WebRTC.

6.1.4 Autenticação do utilizador no Asterisk

A autenticação de um utilizador no servidor Asterisk é feita exclusivamente á conta de um nome de utilizador e uma palavra chave. Os nomes de utilizador são os IstID's enquanto que as palavras chave são geradas aleatoriamente pelo servidor Web quando o utilizador se regista na aplicação.

Para conseguir obter a chave do Asterisk um utilizador precisa de aceder ao servidor Web e requisitá-la através da rota ***/mydata,*** mas para isso é necessário passar primeiro pelo processo de autenticação do servidor.

Infelizmente o servidor Asterisk apenas permite configurar uma password directamente no ficheiro ou uma hash MD5. Escrever passwords directamente em ficheiros é uma das piores práticas que se pode ter quando se lida com passwords e usar uma hash MD5 também não seguro uma vez que com o poder

computacional disponível no 2018 é possível calcular uma password que dê a hash que queremos em tempo útil. Este problema é grave e só pode ser mitigado mudando a password com regularidade, um sistema automático que faça esta tarefa é possível de implementar executando o código que gera a configuração do utilizador de tempos a tempos (por exemplo sempre que o utilizador inicia uma nova sessão com o browser e não existe nenhuma outra activa).

6.2 IVR em Node Vs Asterisk

O servidor Asterisk tal como a maioria dos IP-PBX disponibiliza uma framework para a criação de menus IVR que pode ser usada para criar menus de forma rápida. Infelizmente estes menus estão severamente limitados e usam linguagem própria da aplicação nas configurações. Um menu IVR do Asterisk está limitado no tipo de base de dados com que consegue interagir. Além de estar limitado nas bases de dados que são compatíveis a própria linguagem dificulta a construção de menus complexos uma vez que não existem bibliotecas externas que possam ser usadas para agilizar/simplificar o processo de criação do menu.

Por outro lado, os menus IVR implementados podem usar qualquer base de dados, e pode interagir de forma simples e directa com qualquer serviço Web. Tirando partido destas funcionalidades um sistema IVR escrito em NodeJS consegue oferecer aos seus utilizadores um menu totalmente dinâmico que pode recolher informações de um ou mais serviços web. A disponibilidade de milhares de bibliotecas e frameworks para a linguagem também é um factor que torna um menu IVR construído em JavaScript mais atrativo para os programadores.

6.3 Directório

Os dados dos utilizadores obtidos pelo sistema WebRTC são quase todos obtidos através do sistema fénix usando o protocolo OAuth2.0 a excepção são os números SIP dos utilizadores. Os números SIP dos utilizadores são introduzidos pelos próprios usando a aplicação Web. Isto deve-se ao facto de não ser possível obter os números de telefone de um utilizador usando a API que o sistema fénix disponibiliza para as aplicações de terceiros.

Isto acaba por se tornar um incómodo para o utilizador que tem de adicionar ao sistema os seus números/extensões manualmente e também uma falha de segurança uma vez que a única garantia que aquele número pertence efectivamente ao utilizador é dada pelo próprio utilizador que é parte interessada na operação. A falha não é muito grave pelo que o máximo que pode acontecer é um utilizador ser induzido em erro ligando para uma pessoa achando que está a ligar para outra. Para mitigar o problema o sistema também permite que vários utilizadores tenham o mesmo número SIP associado.

6.4 PBX alternativos

Este sistema WebRTC poderia ter sido desenvolvido utilizando outro PBX como por exemplo o FrreSWITCH. As especificações que levaram á escolha do Asterisk também são cumpridas pelo FrreSWITCH, ambos têm compatibilidade com WebRTC e realizam operações de transcodding. O que pesou mais na escolha foi o conhecimento que já existia em relação ao funcionamento do servidor Asterisk o que facilitou o desenvolvimento do projecto.

6.5 Gestão e manutenção do sistema

Os vários componentes deste sistema foram escritos usando a mesma linguagem de alto nível, JavaScript, isto permite que o programador que fique encarregado de gerir, corrigir e estender as funcionalidades do sistema necessite de saber apenas uma linguagem. O JavaScript do lado cliente (browser) é diferente do lado do servidor (NodeJS) mas as diferenças não são muito significativas.

O ponto mais negativo é o facto do servidor Asterisk fazer parte do sistema, isto porque a configuração de um servidor Asterisk requer conhecimentos sobre o funcionamento do protocolo SIP e também sobre como o próprio servidor está estruturado colocando um grande obstáculo a quem o está a utilizar pela primeira vez. No entanto o projecto foi todo desenvolvido tendo este problema em mente pelo que a filosofia foi tentar ao máximo executar todas as funcionalidades fora do servidor Asterisk de modo a deixar a configuração do Asterisk o mais simples possível. Para criar novos menus IVR no futuro não será necessário mexer muito nas configurações do Asterisk, apenas é necessário associar uma extensão ao novo menu IVR o que é possível de fazer em duas simples linhas no dialplan do Asterisk.

7 Conclusão

Este projecto foi capaz de integrar a nova tecnologia WebRTC com tecnologias VoIP existentes sendo capaz de entregar aos utilizadores um verdadeiro telefone através do browser. O projecto foi também capaz de usar serviços existentes para realizar a autenticação dos utilizadores e obter dados, mediante a autorização expressa do utilizador, entregando um sistema final muito simples do ponto de vista do utilizador.

O sistema é capaz de descobrir os números de um qualquer utilizador através de uma simples pesquisa por nome e iniciar uma chamada para qualquer um deles. O sistema consegue também receber chamadas de diversas fontes que vão desde outros browsers até telefones numa rede POTS. A segurança das comunicações é assegurada pelo uso de encriptação em todas as fases de sinalização e na própria sessão que é criada entre os utilizadores.

Usando o servidor multimédia Kurento e o servidor IVR o sistema é capaz de criar caixas de VoiceMail e menus IVR integrados com a Web permitindo que mensagens gravadas usando telefones SIP possam ser ouvidas num browser e vice-versa. Os menus IVR também podem ser muito mais complexos uma vez que estes são executados como um programa NodeJS oferecendo a possibilidade de integrar estes menus com qualquer outro serviço Web (aceder API's web, aceder a várias bases de dados, etc…).

A aplicação Web mostrou ser compatível com os browsers Chrome e Firefox nas versões desktop e mobile. O browser Edge não é suportado e o Internet Explorer não implementa a tecnologia WebRTC. Apesar do funcionamento só estar garantido para dois browsers estes representam uma grande fatia do mercado de WebBrowsers sendo o Chrome o browser mais utilizado (58,51%) e o Firefox o terceiro mais utilizado (6,71%) durante o mês de fevereiro de 2018 segundo a NetMarketShare.

Dado o cumprimento dos requisitos e a vasta disponibilidade da aplicação em vários dispositivos é possível concluir que o sistema pode ser usado como um substituto a rede POTS caso esta falhe. O facto de não necessitar de instalação e de poder ser usado sem restrições após o primeiro LogIn permite uma rápida mudança de todos os utilizadores para o novo sistema. Usar um softphone implica um processo de configuração mais complicado em que o utilizador tem de instalar e configurar a aplicação em cada um dos seus dispositivos sendo que este processo pode ser complicado para algumas pessoas especialmente se for usada encriptação nas sessões. O facto de a aplicação Web poder ser usada tanto em mobile como em Desktop também ajuda os utilizadores uma vez que a interface é igual em todos os dispositivos.

8 Bibliografia

[1] L. F. M. H. P. C. E. B. Craft, “Machine Switching Telepnhone System for Large Metropolitan Areas,” pp. 53-89, 14 Fevereiro 1923.

[2] “Integrated Services Digital Network,” 18 Agosto 2018. [Online]. Available: https://en.wikipedia.org/wiki/Integrated_Services_Digital_Network. [Acedido em 6 Outubro 2018].

[3] Y. L. A. Uzelac, “Voice over IP (VoIP) SIP Peering Use Cases,” Novembro 2011. [Online]. Available: https://tools.ietf.org/html/rfc3261. [Acedido em 6 Outubro 2018].

[4] H. Schulzrinne, S. Casner, R. Frederick e V. Jacobson, “RFC 3550 - RTP: A Transport Protocol for Real-Time Applications,” Julho 2003. [Online]. Available: https://tools.ietf.org/html/rfc3550. [Acedido em 13 Abril 2018].

[5] M. Baugher, D. McGrew, M. Naslund, E. Carrara e K. Norrman, “RFC 3711 - The Secure Real-time Transport Protocol (SRTP),” IETF, Março 2004. [Online]. Available: https://tools.ietf.org/html/rfc3711. [Acedido em 13 Abril 2018].

[6] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, M. R. Sparks, Handley e E. Schooler, “RFC3261 - SIP: Session Initiation Protocol,” Junho 2002. [Online]. Available: https://tools.ietf.org/html/rfc3261. [Acedido em 13 Abril 2018].

[7] M. Handley, V. Jacobson e C. Perkins, “RFC 4566 - SDP: Session Description Protocol,” Julho 2006. [Online]. Available: https://tools.ietf.org/html/rfc4566. [Acedido em 13 Abril 2018].

[8] Google Chrome team, “WebRTC Home | WebRTC,” [Online]. Available: https://webrtc.org/. [Acedido em 13 Abril 2018].

[9] J. Rosenberg, “RFC 5245 - Interactive Connectivity Establishment (ICE): A Methodology for Network Address Translator (NAT) Traversal for Offer/Answer Protocols,” Abril 2010. [Online]. Available: https://tools.ietf.org/html/rfc5245. [Acedido em 13 Abril 2018].

[10] J. Rosenberg, R. Mahy, P. Matthews e D. Wing, “RFC 5389 - Session Traversal Utilities for NAT (STUN),” Outubro 2008. [Online]. Available: https://tools.ietf.org/html/rfc5389. [Acedido em 26 Abril 2018].

[11] A. Prokop, “Understanding WebRTC Media Connections: ICE, STUN and TURN,” 11 Agosto 2014. [Online]. Available: https://www.avaya.com/blogs/archives/2014/08/understanding-webrtc-media-connections-ice-stun-and-turn.html. [Acedido em 26 Abril 2018].

[12] E. Ivov, E. Rescorla, J. Uberti e P. Saint-Andre, “draft-ietf-ice-trickle-20 - Trickle ICE: Incremental Provisioning of Candidates for the Interactive Connectivity Establishment (ICE) Protocol,” 9 Abril 2018. [Online]. Available: https://tools.ietf.org/html/draft-ietf-ice-trickle-15. [Acedido em 13 Abril 2018].

[13] Digium, “Open Source Communications Software | Asterisk Official Site,” [Online]. Available: https://www.asterisk.org/. [Acedido em 30 Setembro 2018].

[14] FreeSWITCH, “Home - FreeSWITCH,” [Online]. Available: https://freeswitch.com/. [Acedido em 30 Setembro 2018].

[15] “Kurento,” [Online]. Available: http://www.kurento.org/. [Acedido em 10 Outubro 2018].

[16] E. T. Melanchuk, “An Architectural Framework for Media Server Control,” Junho 2009. [Online]. Available: https://tools.ietf.org/html/rfc5567#page-3. [Acedido em 5 Outubro 2018].

[17] “Solutions | OnSIP,” [Online]. Available: https://www.onsip.com/solutions. [Acedido em 21 Abril 2018].

[18] Nexmo Inc., “Nexmo - APIs for SMS, Voice and Phone Verifications,” [Online]. Available: https://www.nexmo.com/. [Acedido em 21 April 2018].

[19] I. B. C. S. I. C. José Luis Millán, “JsSIP: The JavaScript SIP Library,” [Online]. Available: http://jssip.net/. [Acedido em 6 Outubro 2018].

[20] E. G. J. F. W. M. James Criscuolo, “WebRTC Made Easy for JavaScript Developers,” [Online]. Available: https://sipjs.com/. [Acedido em 6 Outubro 2018].

[21] D. Hardt, “RFC 6749 - The OAuth 2.0 Authorization Framework,” Outubro 2012. [Online]. Available: https://tools.ietf.org/html/rfc6749. [Acedido em 2018 Abril 18].

[22] R. Newton e R. Mudgett, “PJSIP Configuration Sections and Relationships,” 5 Fevereiro 2018. [Online]. Available: https://wiki.asterisk.org/wiki/display/AST/PJSIP+Configuration+Sections+and+Relationships. [Acedido em 28 Julho 2018].

[23] “RTCPeerConnection - Web APIs | MDN,” 9 Março 2018. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/RTCPeerConnection. [Acedido em 15 Abril 2018].

[24] “MediaDevices.getUserMedia() - Web APIs | MDN,” 5 Outubro 2017. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices/getUserMedia. [Acedido em 15 Abril 2018].

[25] “MediaRecorder - Web APIs | MDN,” 8 Março 2018. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/MediaRecorder. [Acedido em 15 Abril 2018].

[26] A. Freier, P. Karlton e P. Kocher, “RFC 6101 - The Secure Sockets Layer (SSL) Protocol Version 3.0,” Agosto 2011. [Online]. Available: https://tools.ietf.org/html/rfc6101. [Acedido em 18 Abril 2018].

[27] NTT Communications, “A Study of WebRTC Security,” [Online]. Available: http://webrtc-security.github.io/. [Acedido em 19 Abril 2018].

[28] A. Reginato, “simple-oauth2 - npm,” [Online]. Available: https://www.npmjs.com/package/simple-oauth2. [Acedido em 18 Abril 2018].

[29] E. Rescorla e N. Modadugu, “RFC 6347 - Datagram Transport Layer Security Version 1.2,” Janeiro 2012. [Online]. Available: https://tools.ietf.org/html/rfc6347. [Acedido em 19 Abril 2018].

[30] “WebRTC,” [Online]. Available: http://en.wikipedia.org/wiki/WebRTC. [Acedido em 13 Abril 2018].

[31] F. Andreasen, M. Baugher e D. Wing, “RFC 4568 - Session Description Protocol (SDP) Security Descriptions for Media Streams,” Julho 2006. [Online]. Available: https://tools.ietf.org/html/rfc4568. [Acedido em 23 Abril 2018].

[32] “SIP Signaling JavaScript Library for WebRTC Developers | SIP.js,” [Online]. Available: https://sipjs.com/. [Acedido em 10 Outubro 2018].

9 Anexos

A. Asterisk http.conf

[general]

enabled=yes

bindaddr=0.0.0.0

bindport=8088

tlsenable=yes

tlsbindaddr=0.0.0.0:8089

tlscertfile=/etc/asterisk/keys/asterisk.pem

B. Asterisk rtp.conf

```
[general]
rtpstart=10000
rtpend=20000
icesupport=yes
stunaddr=stun.l.google.com:19302
rtp_symetric=yes
nat=force_rport
```

C. Asterisk pjsip.conf

```
;======================        Transporte        ============================
[transport-wss]
type=transport
protocol=wss
bind=0.0.0.0:8089
[transport-ws]
type=transport
protocol=ws
bind=0.0.0.0:8088
[simple-udp]
type=transport
protocol=udp
bind=0.0.0.0:5061
;================        Templates        =====================
[aor-single-reg](!)
type=aor
max_contacts=10
remove_existing=no

 [auth-userpass](!)
type=auth
auth_type=userpass

[endpoint-webrtc](!)
type=endpoint
context=internal
use_avpf=yes
media_encryption=dtls
dtls_cert_file=/etc/asterisk/keys/asterisk.pem
dtls_verify=fingerprint
dtls_setup=actpass
ice_support=yes
rtcp_mux=yes
dtmf_mode=info
disallow=all
allow=opus

[endpoint-udp](!)
type=endpoint
context=internal
```

```
disallow=all
allow=opus
```

D. DialPlan

```
[macro-sendNumber]
exten => s,1,NoOp()
        same => n,SendDTMF(${CALLERID(num)})

[internal]
exten => _XXXXXX,1,Ringing()
        same => n,Dial(${PJSIP_DIAL_CONTACTS(${EXTEN})},60)
        same => n,Dial(PJSIP/record-ivr,3,M(sendNumber))

exten => 1000,1,Ringing()
        same => n,Dial(PJSIP/voicemail-ivr)
```